Lüers

MSX

Grafik & Sound

EIN DATA BECKER BUCH

Lüers

MSX
Grafik & Sound

EIN DATA BECKER BUCH

ISBN 3-89011-051-7

Copyright © 1985 DATA BECKER GmbH
Merowingerstraße 30
4000 Düsseldorf

Alle Rechte vorbehalten. Kein Teil dieses Buches darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne schriftliche Genehmigung der DATA BECKER GmbH reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Wichtiger Hinweis:
Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.



INHALTSVERZEICHNIS
=================

Sehr geehrte, liebe MSX-Freunde und MSX-Interessenten!

Bisher schauten die Privatleute, die sich nach reiflicher Überlegung zum Kauf eines Home- oder Heimcomputers entschlossen hatten, neidisch auf die größeren Computer, denn unter diesen Gerätefamilien waren schon vor Jahren vorzügliche Standards geschaffen worden. Die Namen 'CP/M' und 'MS-DOS' sowie der Ausdruck 'IBM-Kompatibilität' gewährleisteten und tun dies gerade heute in immer stärkerem Maße, daß die Programmvielfalt von Computer X auch auf Computer Y ohne Probleme läuft und umgekehrt selbstverständlich ebenso. Da gibt es keine Eifersuchtsprobleme nach dem Motto 'Warum kann ich diese Datenbank ausgerechnet nicht auf meinem Computer laufen lassen?!'.

Wie sah und sieht es heute noch bei den meisten Homecomputern aus? Gerätehersteller X bringt einen Computer heraus, zu dem natürlich die Geräteanschlüsse und auch die Programmvielfalt nur dieses einen Herstellers passen. So zieht der Kauf des Computers X im Normalfall auch ein beachtliches Nachfolgegeschäft nur bei diesem Hersteller nach sich, woran sich manche bereits ein goldenes Näschen verdient haben. Ist der Markt mit Produkt X halbwegs gesättigt, bringt der gleiche Hersteller ein Produkt Y heraus. Damit auch hiernach wieder per Nachfolgegeschäft die Kassen klingeln, werden halt neue Anschlüsse (wieder keinem Standard genügend) am Computer angebracht, und - wo käme der arme Hersteller denn sonst auch hin - die Software von Produkt X läuft selbstverständlich nicht mehr auf dem Computer Y. Ob dieser Hersteller nicht irgendwann einmal mit dieser Politik auf die Nase fallen wird ... warten wir es ab ... die MSX-Computer können nicht nur, sie müssen Sieger am Ende der Wegstrecke sein, denn hier wird niemand mehr für 'dumm' verkauft.

Wie kann so etwas gelingen? Alle japanischen Firmen aus dem HIFI-VIDEO-Bereich haben sich vor einiger Zeit gemeinsam an einen großen Tisch gesetzt und beratschlagt, wie den Wirtschaftsmächten USA und Europa mal wieder ein Schnäppchen mit bombastischer Wirkung geschlagen werden könne (siehe auch die Entwicklung bei Kameras, dem HIFI-VIDEO-Sektor und der Automobilproduktion nicht zuletzt).

Da man den Homecomputermarkt nicht als erster betreten würde, mußte nun gemeinsam etwas besonderes ausgegoren werden. Schließlich einigte man sich darauf, Homecomputer auf den Markt zu bringen, die nicht nur Kompatibilität (Austauschbarkeit) bei der Software (Programme) sondern auch bei der Hardware (Anschlüsse zu weiteren Geräten) gewährleisten sollten.

Man stellte all' dieses unter den Tenor 'MSX', eine Abkürzung für das leistungsfähigste BASIC, das ... mal wieder von 'MICROSOFT', einer äußerst renommierten amerikanischen Softwarefirma, stammt: 'Microsoft Super Extended Basic'.

Doch der MSX-Standard umfaßt mehr: Nicht nur das BASIC ist gleich, auch die Computerchips für Sound und Grafik (über deren Fähigkeiten wir uns in diesem Buch sehr ausführlich auseinandersetzen) gleichen bei allen MSX-Computern wie ein Ei dem anderen. Außerdem hat jeder MSX-Computer, soweit zur Zeit zu sehen, eine Standard Centronics Schnittstelle, die Verwendung von kommerziellen CP/M-Programmen (s.o. - eigentlich kommt diese Software ja von den größeren Computern her!) ist zu 100% möglich und last not least: Das MSX-Diskettenbetriebssystem MSX-DOS ist datenkompatibel zum MS-DOS von IBM ... d.h. Sie können sich Ihre Daten aus dem Büro nach Hause mitnehmen (Vorsicht! Datenschutz!) und in aller Ruhe auf Ihrem MSX-Computer mit angeschlossener Diskettenstation weiterverarbeiten!!!

Sie merken vielleicht, eine Überlegung der Japaner, die Hand und Fuß hat. Diese Idee muß einfach erfolgreich sein, anders kann es gar nicht sein!

Zwei Punkte seien noch angefügt:

1) Weil die Hersteller von MSX-Computern fast allesamt aus der HIFI-VIDEO-Branche stammen, ist der Anschluß des MSX-Homecomputers z.B. an den Videorecorder, den Bildplattenspieler, die Tonbandmaschine usw. gar nicht mehr fern (Beispiele dafür gab es bereits 1984 auf der HIFIvideo in Düsseldorf en masse zu sehen). Also: ein Homecomputersystem, das wahrlich ins häusliche Innenleben hereingehört, ja, sich sogar in seine Umgebung schnellstens einpaßt.

2) Nicht nur die Anwendungsmöglichkeiten der MSX-Computer bestechen! Auch technische Daten wie Ausbaubarkeit auf 256 KB RAM, 32 Sprites, 16 Farben, hochauflösende Farbgrafik mit 256*192 Punkten und ein eingebauter dreistimmiger Soundprozessor sprechen für den Computerkenner geradezu Bände!

Warten wir ab, wie sich der Markt umkrempelt. Die Japaner (repräsentiert durch mehr als zwanzig Firmen mit renommierten Namen wie SONY, SPECTRAVIDEO, PANASONIC, JVC usw.), die Koreaner (u.a. GOLDSTAR und SAMSUNG) und auch PHILIPS aus Deutschland erhoffen, daß berteits 1985 ein Großteil aller verkauften Homecomputer das Kürzel MSX tragen werden. Machen wir es dem europäischen Ausland nach! Toi, Toi, Toi MSX!

P.S.: Die Begriffe 'Homecomputer' und 'Personalcomputer' sind inzwischen so nichtssagend, daß wir bei allen MSX-Computern in jedem Fall auch von Personalcomputern sprechen können.



Sehr geehrte liebe Leserin!
Sehr geehrter lieber Leser!
Sehr geehrte liebe Programmiererin!
Sehr geehrter lieber Programmierer!

Lange Rede - kurzer Sinn: Ich heiße Sie herzlich willkommen im großen (internationalen) Kreis der MSX-User und natürlich als stolzer und neugieriger Besitzer dieses DATA BECKER-Buchs.

Auf der Buchrückseite und im Inhaltsverzeichnis konnten Sie bereits erfahren, was Sie in diesem Buch erwartet. Hier und an dieser Stelle nur soviel:

Es war nicht leicht ein Buch zu schreiben, mit dem sowohl der Anfänger als auch der Profi etwas anfangen kann. So habe ich versucht, den goldenen Mittelweg zu finden z.B. werden bei fast allen MSX-Befehlsbeschreibungen Hinweise zu den entsprechenden Systemvariablen gegeben - für den Anfänger nur interessant - für den Profi aber lebensnotwendig.

Hoffentlich finden Sie Gefallen am Inhalt dieses Buches. Nicht nur das! Vielleicht liegt dieses Buch bald immer neben Ihrem MSX-Computer und entwickelt sich gar zu einem nützlichen Nachschlagwerk.

Wie Sie dieses Buch auch benutzen, auf jeden Fall wünsche ich Ihnen viel Freude dabei. Haben Sie Fragen oder wollen mir etwas persönlich mitteilen, so rufen Sie bitte 0251/47478 an. Dort steht ein Anrufbeantworter und nimmt Ihre Telefonate gerne entgegen. Geben Sie bitte Ihre vollständige Adresse oder im Notfall auch Ihre Telefonnummer an. Ich versuche, Ihnen so schnell wie irgend möglich Nachricht zukommen zu lassen.

Mit freundlichen Grüßen

der Autor dieses DATA BECKER Buchs

P.S.: Ein Hinweis zu den Programmen in diesem Buch:
In den Erklärungszeilen (REM) sind auch deutsche Umlaute enthalten. Haben Sie Probleme, die Umlaute mit Ihrem MSX-Computer einzugeben, lassen Sie die REM-Zeilen einfach weg. Diese Zeilen sind eh nur zu Ihrer Information in die Programme eingebaut worden.

Befehle Grafik SCREEN

Änderung des SCREENs und verschiedener Ein- Ausgabevorgänge

siehe hierzu auch die Programme im Anhang: SCREEN 1
SCREEN 2
SCREEN 3
SCREEN 4
SCREEN 5
SCREEN 6
SCREEN 7
SCREEN 8
SCREEN 9
SCREEN 10

Der Befehl SCREEN ist in gewisser Weise ein Zentralkommando des MSX-Basic, denn durch ihn kann nicht nur zwischen verschiedenen Text- und Grafikbildschirmen hin- und hergewechselt werden, nein, durch SCREEN wird die an Ihren MSX-Computer angeschlossene Peripherie vielseitig und differenziert nutzbar gemacht.

Man kann den SCREEN-Befehl samt seiner möglichen Parameteranzahl vielseitig nutzen: So ist es möglich, nur die Anweisungen hervorzurufen, die einem sinnvoll erscheinen, während die anderen Anweisungen (vorherige Parameter) durch die Kommaeingabe übersprungen werden.

Der Aufbau des SCREEN-Kommandos lautet im einzelnen wie folgt:

```
SCREEN A,B,C,D,E
```

Kommen wir nun im einzelnen auf die verschiedenen Parameter und ihre Wirkungsweise zu sprechen:

zu A: Dieser Parameter dürfte die größte Bedeutung des SCREEN-Kommandos ausmachen, denn hiermit wird zwischen den verschiedenen möglichen 'Screens', ob Text oder Grafik, umgeschaltet.

Parameter A kann die Werte 0 bis 3 annehmen, wobei 0 und 1 vorwiegend zur Textdarstellung dienen, während 2 und 3 hauptsächlich für die Darstellung von Grafik verantwortlich:

SCREEN 0: nur Text, 1 bis 40 Zeichen in 24 Zeilen
SCREEN 1: Text in 24 Zeilen zu 1 bis 32 Zeichen, Sprites darstellbar
SCREEN 2: hochauflösende Grafik mit 256*192 Punkten, Sprites darstellbar
SCREEN 3: Grafikdarstellung mit 64*48 Punkten möglich, Sprites darstellbar

Soviel zur Unterscheidung von Text- (SCREEN 0 und SCREEN 1) und Grafik- (SCREEN 2 und SCREEN 3)-Bildschirmen, die mit Hilfe des SCREEN-Kommandos unter Hinzuziehung des ersten Parameters anwählbar sind.

Es steht uns jedoch offen, sowohl auf den Textbildschirm Grafik als auch auf den Grafikbildschirm Text darzustellen. Da dieses Verfahren nicht ohne weiteres einsichtig erscheint, gehen wir an dieser Stelle darauf etwas näher ein:

a) Grafik auf den Textbildschirmen:

Wir können uns mit Hilfe der zur Verfügng stehenden Grafikzeichen (z.B. Rechtecke) mit etwas Phantasie kleine Figuren zeichnen. Um uns der Vielfalt der Grafikzeichen bewußt zu werden, können Sie einerseits den Weg wählen, die GRAPH-Taste mit jeweils einer beliebigen Buchstabentaste gleichzeitig zu drücken oder aber die Grafikzeichen mit Hile der CHR$-Funktion auf den Bildschirm zu bringen. Da die letztgenanne Möglichkeit des öfteren in den Programmen dieses Buches Anwendung findet (der hundertprozentigen Identifizierung der Grafikzeichen wegen), wollen wir uns die MSX-Grafikzeichen etwas genauer anschauen:

siehe Kurzlisting SCREEN 1 im Anhang

Wir sind aber nicht dazu verpflichtet, lediglich mit diesem beschränkten Grafikzeichensatz vorlieb zu nehmen. Vielmehr können Sie jedes Buchstaben- oder Zahlzeichen oder ..., sprich alle

nur igendwie bildlich darstelbaren Zeichen definieren. Hierzu bedienen Sie sich ganz einfach der BASE-Adresse im Speicher, die uns den Beginn des gespeicherten Charakteraussehens verrät:

```
PRINT BASE(2)

Ergebnis: 2048
```

Da sich jedes definierte Zeichen aus 8 Zeilen zusammensetzt, müssen wir einen Trick anwenden, um z.B. das Leerzeichen (CHR$(32)), das im Normalfall Großteile der Bildschirmfläche überdeckt, im Aussehen zu ändern:

```
PRINT VPEEK(BASE(2)+(32*8)+1)
```

Mit dem Kommando VPEEK schauen wir uns den Video-RAM-Speicher an der uns zur Zeit sehr interessierenden Stelle etwas genauer an. Der Wert 0 verrät uns, daß sich das Leerzeichen aus zumindest einer leeren Zeile (=0) zusammensetzt. Was würde wohl passieren, wenn wir diese Speicherstelle zu ändern versuchten?

```
VPOKE BASE(21),(32*8)+1,255
```

Falls Sie den Befehl gleich eingegeben haben, ist Ihnen folgendes keine Überraschung mehr: Sofort nach Ausführung dieser Anweisung verändert sich ein Großteil des Bildschirms sofort. Was wir getan haben? Wir beauftragten unseren "Kollegen MSX-Computer" dazu, die Speicherstelle, die zum Leerzeichen gehört, mit 255 = einem weißen Balken zu belegen.

Da auf den Textbildschirmen aber 'nur' 256 verschiedene Zeichen darstellbar sind, veränderte sich jedes Leerzeichen (eins der 256 Zeichen) augenblicklich gemäß der neuen = veränderten Codeeingabe.

Sie können diesen beim weiteren Arbeiten störenden Effekt hinwegzaubern, indem Sie an die geänderte Speicherstelle wieder mit VPOKE den alten Wert eingeben. Sie können aber auch alle ursprünglichen Buchstaben und Zeichen zu jedem Zeitpunkt neu laden, indem Sie irgendeinen SCREEN-Bildschirm und dem ersten Parameter eingeben.

Auf diese gerade beschriebene Art und Weise ist es aber kaum möglich - vorrausgesetzt wir verzichten auf die gleichzeitige Text- und Grafikdarstellung auf SCREEN 0 bzw. SCREEN 1 vollständig - auf einen kleinen Teil des Textbildschirms (insgesamt eine Fläche von bis zu 256*192 Punkten) Bilder zu malen. Zufällig erzeugt sei dies im folgenden Programm gezeigt:

siehe Kurzlisting SCREEN 2 im Anhang

Geben Sie nach dieser eindrucksvollen Bildschirmänderung wieder SCREEN 0 ein und alles hat sich wieder zum positiven geändert (Sie können die SCREEN-Eingabe leider in diesem Fall nicht auf dem Bildschirm erkennen, deshalb geben Sie blind ein und drücken anschließend die Tastenkombination CTRL-E (zum Löschen bis zum Zeilenende)).

Diese eingeschränkte Grafikdarstellung auf dem Textbildschirm SCREEn 0 ist Ihnen hoffentlich deutlich geworden; jedoch soll der Hinweis an dieser Stelle nicht fehlen, daß Sie es vermeiden sollten, direkte Grafikkommandos (wie CIRCLE, PAINT, LINE) aufzurufen, während Sie sich auf den Textbildschirmen befinden. Die Errormeldung Illegal function call wird sonst das Resultat sein.

b) Text auf den Grafikbildschirmen

Wie schaut es nun mit Text auf den Grafikbildschirmen aus? Eine einfache PRINT-Anweisung genügt bei den MSX-Computern leider noch nicht:

siehe Kurzlisting SCREEN 3 im Anhang

Vielmehr müssen Sie zum Textschreiben auf den Grafikbildschirmen vorher erst einmal einen Datenkanal öffnen (mit OPEN) und daraufhin die PRINT #-Anweisung direkt an diesen Kanal schicken, wie dies das folgende Programm demonstriert:

siehe Kurzlisting SCREEN 4 im Anhang

Bei Programm SCREEN 3 geben Sie in Zeile 30 Ihrem MSX-Computer Bescheid, über welche Datenleitung die Ausgabe auf den Grafikbildschirm erfolgen soll. Hiernach haben Sie allerdings - da grafikmäßig = Punkt für Punkt erzeugt - die Möglichkeit, einen zweiten Text oder gar eine Grafik über das erste Schriftbild zu drucken. Probleme können dabei nur auftreten, wenn Sie mit unterschiedlichen Farben gleichzeitig arbeiten wollen.

Mit dem DRAW-Kommando ist es auf den Grafikbildschirmen möglich, ähnlich dem LOCATE-Befehl zur Positonierung des Cursors auf dem Textildschirm, - den Text pixelweise dorthin zu positionieren, wo er uns besonders sinnvoll erscheint z.B. zeichnen wir im folgenden Programm die Buchstabenkombination 'MSX' in Form einer Sinuskurve mehrmals auf den Bildschirm:

siehe Kurzlisting SCREEN 5 im Anhang

SCREEN 3 bietet uns ebenfalls wie SCREEN 2 die Möglichkeit, sowohl farbige Grafik als auch Text abzubilden. Besondere Vorsicht gebührt allerdings in diesem SCREEN der richtigen Einschätzung der Größe von abgebildeten Buchstaben und Zeichen wegen. Die Koordinatenansprache hingegen verläuft gleich wie in SCREEN 2.

Noch ein Hinweis:

Besonders aufmerksame Leser, die den DRAW-Befehl bereits kennen, werden sich zwei Fragen stellen, die hier kurz beantwortet werden sollen:

Können wir die Buchstaben und Zeichen mit Hilfe des DRAW "S"-Befehls auf dem Bildschirm in größerem oder kleinerem Maßstab darstellen?

Antwort: Nein, dazu bedarf es eines neu definierten Zeichensatzes mit Hilfe der DRAW-Anweisung, denn der DRAW-Befehl im obigen Programm bezieht sich nur auf die genaue Positionierung des Grafikcursors, nicht aber auf die Form der ausgegebenen Buchstaben und Zeichen.

Es gibt aber dennoch eine Möglichkeit, den Text vergrößert und/oder verdreht auf den Grafikbildschirmen darzustellen. Wir lesen zu diesem Zweck die Bildschirmpunkte mit POINT vom Bildschirm und geben mit PSET die veränderte Darstellung gleich wieder aus.

siehe Kurzlisting SCREEN 6 im Anhang

2) Da wir mit dem Kommando SCREEN 0 auf den Standardzeichensatz umstellen, müßte demnach das Zeichenaussehen noch im Speicher (ROM) abgelegt sein.

Antwort: Richtig. Schauen Sie sich zu diesem Zweck mit folgendem Programm den entsprechenden ROM-Ausschnitt in Binärdarstellung einmal an:

siehe Kurzlisting SCREEN 7 im Anhang

Noch einmal derselbe Zeichensatz (kopiert aus obigen Speicherstellen im ROM) befindet sich zudem bei Verwendung von SCREEN 0 im Video-RAM-Speicher:

siehe Kurzlisting SCREEN 8 im Anhang

Die Textdarstellung auf Grafikbildschirm SCREEN 2 ähnelt nur rein äußerlich der Wiedergabe auf den Textbildschirmen: Einerseits können wir auf der Grafikseite die Anzahl der Zeichen pro Bildschirmzeile nicht vorbestimmen (auf Textbildschirmen mit WIDTH möglich), wir haben immer 32 Zeichen/Zeile, wenn nicht Buchstabe für Buchstabe mit DRAW "B M" gesetzt wird, andererseits scrollt der Bildschirm nicht, wenn Zeile 24 voll beschrieben wurde. Vielmehr beginnt Ihr MSX-Computer zu diesem Zeitpunkt wieder oben zu schreiben d.h. den bereits dort stehenden Text nicht zu löschen, sondern zu überschreiben. Außerdem können wir

nur auf komplizierte Art und Weise den Text wieder auf dem Grafikbildschirm löschen, wollen wir nicht den Gesamtbildschirm löschen: Wir können entweder ein Rechteck in der Größe des zu löschenden Textes zeichnen und dies mit der Untergrundarbe ausmalen (Funktion LINE ...,,BF) oder wir können den gleichen Text noch einmal genau an derselben Bildschirmposition schreiben, diesmal allerdings in der Untergrundarbe; oder schließlich können wir versuchen, durch VPOKEs im Video-RAM Adresse für Adresse den gewünschten Textausschnitt wieder auf Untergrundfarbe zu setzen (dies dürfte bei weitem der komplizierteste Weg sein).

Schauen Sie sich abschließend zur Erklärung von Parameter 1 des SCREEN-Befehls einmal an, wie mit Hilfe eines ausgefüllten Rechtecks gelöscht, daraufhin geschrieben, dann wieder gelöscht ... werden kann. Das Löschen und Wieder-Hinschreiben erzeugt schließlich einen Blinkeffekt:

siehe Kurzlisting SCREEN 9 im Anhang

Kommen wir nun zu den weiteren Parametern des SCREEN-Kommandos zu sprechen:

zu B: Wollen wir diesen Parameter alleine durch den SCREEN-Befehl ansprechen, müssen wir für den nicht angesprochenen ersten Parameter ein Komma eingeben, z.B.:

```
SCREEN ,1
```

Mit Parameter B geben wir an, wie Sprites auf dem Bildschirm auszusehen haben. Wir haben hierzu vier Möglichkeiten:

SCREEN ,0: Spritegröße 8*8, Originalpixelgröße
SCREEN ,1: Spritegröße 8*8, vergrößerte Pixeldarstell.
SCREEN ,2: Spritegröße 16*16, Originalpixelgröße
SCREEN ,3: Spritegröße 16*16, vergrößerte Pixeldarst.

Da immer nur eine Spritegröße auf dem Bildschirm für sämtliche definierten Sprites zur Verfügung steht, ist es dringendst erforderlich, daß Sie sich bei Ihrer Programmplanung eingehend darüber Gedanken machen, in welcher Form die Sprites auf dem Bildschirm erscheinen sollen (8*8- oder 16*16-Format, vergrößert oder normal dargestellt).

Damit Sie die Möglichkeit haben, dies eingehend zu untersuchen, brauchen Sie mit Hilfe des Kommandos (bzw. der Variablenzuweisung) SPRITE$ nur von vornherein ein 16*16-Sprite zu definieren. Was Ihr MSX-Computer in der von Ihnen gewählten Spritegrößendarstellung dann nicht verwenden kann z.B. wird bei Anwahl von SCREEN,0 nur die linke obere Ecke des eigentlichen 16*16-Sprites gezeigt, das wird einfach nicht auf dem Bildschirm gezeigt.

Durch die Umbelegung der einen auf die andere Spritegröße werden alle aktuellen Spritedarstellungen auf dem Bildschirm und sämtliche Spritedefinitionen im Speicher (VRAM) gelöscht, da nun die Aufteilung des Spritespeichers umorganisiert werden muß (dies tut Ihr MSX-Computer selbstverständlich alleine von sich aus).

Folgendes Programm zeigt Ihnen einmal die verschiedenen Spritegrößen nacheinander:

siehe Kurzlisting SCREEN 10 im Anhang

Anzumerken bleibt vielleicht noch (wie auch unter dem SPRITE$-Kommando bereits angedeutet), daß Sie wohl auf dem Bildschirm bei jedbeliebiger Spritegrößenstufe bis zu 32 verschiedene Sprites gleichzeitig auf dem Bildschirm darstellen können. Verwenden Sie jedoch die kleineren Sprites (8*8 bzw. 8*8 vergrößert), können Sie im Speicher gleichzeitig bis zu 256 verschiedene Spriteaussehen ablegen. Gebrauchen Sie die größeren Sprites (die in Wirklichkeit aus 2*2=4 kleinen 8*8-Sprites zusammengebaut sind) mit 16*16 Einzelpunkten, können Sie im festumgrenzten Sprite Generator Table des VRAM "nur" 64 verschiedene Spriteaussehens abrufbereit niederlegen.

zu C: Damit Sie wie bei großen Personalcomputern auch mit Ihrem MSX-Computer nicht nur gefühlsmäßig und visuell, sondern auch auditiv eine Kontrolle darüber haben, inwieweit irgendeine Taste des Computers gedrückt wurde, ertönt bei jedem Tastendruck ein leises Clicken (bei auftretenden Fehler ertönt hingegen ein lauttönendes BEEP).

Sollte Ihnen dieses andauernde Clicken auf den Wecker gehen, können Sie es durch Eingabe von

```
SCREEN ,,0
```

(wir überspringen in diesem Fall die ersten zwei SCREEN-Parameter) beenden. Ein Einschalten ist selbstverständlich ebenfalls jederzeit möglich, indem Sie diesen Parameter Nummer 3 mit einer Zahl zwischen 1 und 255 belegen, wie folgendes Beispiel zeigt:

```
SCREEN ,,1 (Tastaturclicken wieder an)
SCREEN ,,0 (Tastaturclicken wieder aus)
```

zu D: Haben Sie Ihre ersten Programmierschritte hinter sich gebracht und wollen Ihr erzieltes Ergebnis (zumindest ein einzeiliges Programm) abspeichern, so können Sie dies sowohl auf einem Kassettenrecorder (ein x-beliebiger, der sich allerdings mit den passenden Kabelanschlüssen zum MSX-Computer vertragen muß) als auch mit einer Diskettenstation (ein kleiner Plattenspieler, dern allerdings nicht mit einem Saphier sondern mit einer Art von Tonkopf Daten magnetisch lesen und/oder speichern kann) durchführen.

Die Kassettenladung bzw. -speicherung nimmt bei Ihrem MSX-Computer einige Zeit in Anspruch (bei einem längeren Spielprogramm, wie hinten im Anhang abgedruckt, sind dies gar mehrere Minuten). Sie können zwei verschiedene Wege gehen: Entweder Sie haben einen sehr guten Recorder und verwenden zur Abspeicherung/Ladung von Programmen ein gutes Ferrokassettenband; in diesem Fall geben Sie mit

```
SCREEN ,,,2
```

Ihrem MSX-Computer darüber bescheid, daß er mit doppelter Geschwindigkeit (d.h. 2400 Baud = 2400 Bits pro Sekunde = ungefähr 300 Buchstaben oder Zeichen pro Sekunde) abspeichern soll; trifft dies nicht zu (Sie dürfen dann allerdings auch nicht unbedingt die schlechtesten Recorder respektive Tonbandkassetten zum Laden/Speichern verwenden), so geben Sie vor dem Abspeichern/Laden das BASIC-Kommando

```
SCREEN ,,,1
```

ein. Nun werden die Daten mit normaler Geschwindigkeit auf Kassetten gespeichert (d.h. 1200 Baud = 1200 Bits pro Sekunde = ungefähr 150 Buchstaben oder Zeichen pro Sekunde).

Wie geht das vor sich? Die Daten fließen halt ganz einfach gesagt in doppelter Geschwindigkeit zum Kassettenrecorder hin, nicht, daß Sie vermuten, Ihr einfacher Kassettenrecorder müßte nun auf Turbogeschwindigkeit getunt werden!

Schließlich noch zum letzten Parameter, den uns das SCREEN-Kommando anbietet:

zu E: Es gibt seit Jahren eine inoffizielle Abmachung zwischen den verschiedenen Computerherstellern darüber, daß eine Kompatibilität aller Computersystemen angepeilt werden sollte. Dabei haben sich die Firmen ASCII (für den Zeichensatz) und MICROSOFT (für das BASIC) bisher die verdienstvollsten Namen auf diesemem heiß umkämpften Markt der Eigenbrödler erworben.

Ihr MSX-Computer verfügt nun noch über eine Stufe mehr: Sein BASIC ist kompatibel (100 Prozent) zu allen anderen MSX-Computern, aber nicht nur das: Die Peripherieanschlüsse sind gleich ... einfach fast alles ist gleich, bis auf wenige Ausnahmen.

Was hat das mit der Erläuterung von Parameter E zu tun? Die Buchstaben und Zeichen 32 bis 127 Ihres MSX-Computers entsprechen 100 prozentig der ASCII-Norm. Diese Zeichen (alle Buchstaben, Zahlen und Text-Sonderzeichen) können sowohl von einem

speziellen MSX-Drucker als auch von jedem beliebigen Standarddrucker ohne irgendwelche Probleme (eine passende Kabelverbindung selbstverständlich vorausgesetzt) zu Papier gebracht werden.

Darüberhinaus (ab Zeichen 128) verfügt Ihr MSX-Computer aber noch zusätzlich über mnannigfaltige Sonderzeichen, Grafikzeichen und z.B. deutsche Umlaute. Ist ein MSX-Drucker an Ihren MSX-Computer angeschlossen, gibt es auch beim Ausdruck dieser Zeichen keine Probleme, vorausgesetzt, Sie haben Parameter E beim SCREEN-Befehl mit 0 besetzt:

```
SCREEN ,,,,0
```

Haben Sie einen Nicht-MSX-Drucker angeschlossen, müssen die Sonderzeichen ab CHR$(128) zwecks Vermeidung von Kauderwelsch auf dem Drucker als Leerzeichen ausgegeben werden. Ihre Eingabe des SCREEN-Befehls samt E-Parameter müßte dann anstelle der 0 einen größeren Wert (1 bis 255) vorweisen, schon erscheinen alternativ zu den nicht vorhandenen Sonderzeichen Leerzeichen und ... Sie können, falls gewünscht, mit dem Einmalen per Hand beginnen.

SCREEN 1

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 1
30 FOR N=192 TO 255
40 PRINT CHR$(N);
50 NEXT N
```

SCREEN 2

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 0
30 FOR N=0 TO 959
40 PRINT CHR$(INT(RND(1)*223)+32);
50 NEXT N
60 FOR N=2048 TO 4095
70 VPOKE N,INT(RND(1)*255)
80 NEXT N
```

SCREEN 3

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 PRINT "MSX-Computer"
40 GOTO 40
```

SCREEN 4

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 PRINT #1,"MSX-Computer"
50 CLOSE
60 GOTO 60
```

SCREEN 5

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 FOR N=1 TO 255 STEP 3
50 Y=SIN(N/20)*50+100
60 DRAW "bm =n;,=y;"
70 PRINT #1,"MSX";
80 NEXT N
90 CLOSE
100 GOTO 100
```

SCREEN 6

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 DRAW "bm 0,0"
50 PRINT #1,"MSX"
60 FOR N=0 TO 23
70 FOR M=0 TO 7
80 A=POINT(N,M)
90 PSET(N+50,M+50+Z),A
```

```
100 Z=Z+1
110 PSET(N+50,M+50+Z),A
120 NEXT M
130 Z=0:NEXT N
140 GOTO 140
```

SCREEN 7

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 0
30 WIDTH 40
40 FOR N=&H1BBF TO &H1BBF+2047
50 PRINT STRING$(8-LEN(BIN$(PEEK(N)
)),"0 ");BIN$(PEEK(N))
60 NEXT N
```

SCREEN 8

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 0
30 WIDTH 40
40 FOR N=2048 TO 2048+2047
50 PRINT STRING$(8-LEN(BIN$(VPEEK(N))),"
0");BIN$(VPEEK(N))
60 NEXT N
```

SCREEN 9

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 DRAW "bm 128,96"
50 PRINT #1,"MSX"
60 FOR N=1 TO 200
70 NEXT N
80 LINE(128,96)-(152,104),15,BF
90 FOR N=1 TO 200
100 NEXT N
110 GOTO 40
```

SCREEN 10

-*-*-*-*-*-

```
10 COLOR 1,15
20 OPEN "GRP:" FOR OUTPUT AS #1
30 FOR N=0 TO 3
40 SCREEN 2,N
50 DRAW "bm 50,50"
60 SPRITE$(1)=STRING$(32,"U")
70 PRINT #1,"Groesse";N
80 PUT SPRITE 1,(128,96)
90 FOR M=1 TO 1000
100 NEXT M
110 NEXT N
120 GOTO 30
```

Befehle Grafik OPEN

Öffnen von Datenkanälen u.a. zur Textausgabe auf Grafikseiten

siehe hierzu auch die Programme im Anhang: OPEN 1
OPEN 2
OPEN 3
OPEN 4
OPEN 5
OPEN 6
OPEN 7
OPEN 8
OPEN 9
OPEN 10

Mit Hilfe des OPEN-Befehls können wir ganz allgemein gesprochen Datenkanäle öffnen, sei es, daß die Aus-/Eingabe anschließend auf/von Kassettenrecorder, Drucker, Textbildschirm, Grafikbildschirm oder aber einer angeschlossenen Diskettenstation stattfindet. Die Abkürzungen für das gewünschte Aus-/Eingabemedium (sie erfolgt Zeichen nach Zeichen, also sequentiell) lauten:

CAS: für Kassettenreorder
LPT: für Drucker (Line-PrinTer)
CRT: für Textbildschirm
GRP: für Text auf Grafikbildschirm
1: bzw. 2: für Diskettenstationen

Da wir uns mit den Themen Grafik und Sound beschäftigen, wollen wir uns nur mit den diesbezüglichen Ausgabemedien beschäftigen und zwar soweit sie den Drucker, den Textbildschirm und den Grafikbildschirm betreffen:

1) Die Ausgabe mit OPEN "LPT:" auf dem Drucker bzw. OPEN "CRT:" auf dem Textbildschirm:

Diese OPEN-Anweisung wird von Ihnen in den seltensten Fällen im direkten Modus benutzt werden, denn wir verfügen zu diesem Zweck über sehr viel luxuriösere MSX-BASIC-Anweisungen:

PRINT "..." für Ausgabe von Text auf dem Bildschirm
LPRINT "..." für Ausgabe von Text auf den angeschlossenen Drucker
LIST zur Ausgabe von Programmlistings auf dem Bildschirm
LLIST zur Ausgabe des Programmlistings auf dem angeschlossenen Drucker

Der Anschaulichkeit halber sei aber gezeigt, wie die Ausgabe auf dem angeschlossenen Drucker mit dem OPEN-Befehl vor sich zu gehen hat:

```
OPEN "LPT:" FOR OUTPUT AS #1
PRINT #1, "MSX-Computer sind SUPER"
CLOSE #1
```

und die gleiche Ausgabe mit Hilfe des MSX-BASIC-Kommandos LPRINT:

```
LPRINT "MSX-Computer sind SUPER"
```

Sie können sich nun selbst überlegen, welche Arbeit einfacher auszuführen ist: die Awendung des Befehlsspektrums OPEN, PRINT # und CLOSE oder das Kommando LPRINT?!

Warum, so werden Sie fragen, gibt es denn überhaupt die Ausgabemöglichkeit mit OPEN "LPT:", PRINT # und CLOSE? Ganz einfach: wir können auf diese Art und Weise die Ausgabe einmal zu Kanal 1 hinsteuern (im folgenden Beispiel der Textbildschirm), ein anderes Mal zu Kanal 2 (im folgenden Beispiel der Drucker):

siehe Kurzlisting OPEN 1 im Anhang

Auch wäre es möglich, die Ausgabe auf die gleiche Art und Weise noch an zusätzlich zu öffnende Kanäle (3 und 4) zu senden und zwar sinnvollerweise an CAS: (Kassettenrecorder) oder 1: (Diskettenstation) sowie auf den Grafikbildschirm (GRP:). Da unsere Ausgabegeräte samt und sonders mit der nackten Zahl (1, 2, 3 oder 4) angesprochen werden können, ist es somit leicht möglich, die gesamte Ausgabe in einem Programm an Stelle auf dem Bildschirm nun auf den Drucker auszugeben. Welch eine Zeit - und schließlich auch Platzersparnis. Was das bedeutet, werden Sie sich klar machen können, wenn Sie in einem etwas längeren Programm einmal sämtliche PRINT- in LPRINT-Befehle umgewandelt haben (mit der OPEN-Anweisung brauchen sie anstattdessen nur einen Variablenwert zu ändern).

Noch ein kleiner Unterschied zwischen Text- (CRT:) bzw. Drucker (LPT:) -ausgabe im Vergleich zu den BASIC-Kommandos PRINT und LPRINT: Wenn Sie formatiert ausgeben wollen (Spaltenschreibweise zum Erstellen von Tabellen), können Sie hinter den PRINT- bzw. LPRINT-Befehl ein Komma hängen. Somit wird alle 14 Stellen ein Tabulator gesetzt und erst dort weiergeschrieben. Ist die Anzahl der auszugebenden Zeichen größer als die in der Zeile noch darzustellenden Zeichen ohne Überlauf, wird in der nächsten Zeile an der ersten TABulatorzone weiter ausgegeben. Bei Verwendung des OPEN "LPT:" bzw. OPEN "CRT:" hingegen, wird nach einer Kommastelle immer 14 Zeichen später die nächste Ausgabe erzeugt, egal, ob die Anzahl der auszugebenden Zeichen über den Bildschirmrand hinausschießt oder nicht (bei OPEN "LPT:" ist das zudem noch etwas anders):

Ausgabe auf dem Bildschirm mit PRINT:

siehe Kurzlisting OPEN 2 im Anhang

Ausgabe auf dem Bildschirm mit OPEN "CRT:":

siehe Kurzlisting OPEN 3 im Anhang

Ausgabe auf dem Drucker mit LPRINT:

siehe Kurzlisting OPEN 4 im Anhang

Ausgabe auf den Drucker mit OPEN "LPT:":

siehe Kurzlisting OPEN 5 im Anhang

Merken Sie sich deshalb: Mit Hilfe des OPEN-Befehls (Kanal öffnen für Bildschirm- bzw. Druckeransteuerung) ist keine formatierte Ausgabe möglich!

2) Während die Ausgabe auf Drucker bzw. Textbildschirm mit OPEN "CRT:" bzw. OPEN "LPT:" im Gegensatz zu den BASIC-Kommandos LPRINT und LLIST keine schwerwiegenden Unterschiede mit sich bringen, ist dies bei der Ausgabe auf einen der Grafikbildschirme ganz anders. Zu diesem Zweck wollen wir einmal versuchen, Text mit dem PRINT-Befehl ohne mit OPEN einen Datenkanal zu öffnen auf dem Grafikbildschirm auszugeben:

siehe Kurzlisting OPEN 6 im Anhang

So schön und einleuchtend unser kurzes Programm auch ausschaut, nichts tut sich auf dem Grafikbildschirm. Probieren wir es deshalb gleich einmal mit OPEN "GRP:" aus:

siehe Kurzlisting OPEN 7 im Anhang

Nun klappt die Ausgabe einwandfrei und oben links in der Ecke ist unser Schriftzug eindeutig zu erkennen.

Wie bereits an anderer Stelle in diesem Buch erwähnt, müssen wir aber bei der Arbeit auf dem Grafikbildschirm besondere Vorsicht walten lassen, denn hier läuft alles nach anderen Grundsätzen ab, als auf den Textbildschirmen: Im Fall unserer Textausgabe ist es deshab vielleicht wichtig zu wissen, daß Buchstabe für Buchstabe eigentlich als eine Zeichnung von jeweils 8*8 Punkten zu sehen und zu verstehen ist. Aus diesem Grund können wir einerseits Zeile für Zeile mit normalem Zeilen-/Zeichenabstand eingeben (wie in SCREEN 1) und haben dabei 32*24 Zeichen zur Verfügung:

siehe Kurzlisting OPEN 8 im Anhang

oder wir setzen unsere Schrift genau dorthin (und zwar punktweise wählen wir die Bildschirmposition an), wo wir dies wünschen (nicht mehr angewiesen auf das 32*24-Matrixfeld):

siehe Kurzlisting OPEN 9 im Anhang

Da wir auf den Grafikbildschirmen nicht über den LOCATE-Befehl verfügen, müssen wir uns des DRAW "B M"-Kommandos (gehe blind mit dem Cursor an die entsprechende Bildschirmposition) bedienen. Zur Farbänderung des ausgegebenen Textes müssen wir trotz des luxuriösen DRAW-Kommandos entsprechend COLOR eingeben.

Aber bitte schrauben Sie Ihre Erwartungen nicht zu hoch! Die ausgegebenen Zeichen lassen sich auf so einfache Weise leider weder drehen (DRAW "A") noch vergrößerter (DRAW "S") darstellen.

Noch ein Hinweis: Der Grafikbildschirm arbeitet, wie schon gesagt und auch in Programmen bewiesen, ganz anders als die Textbildschirme SCREEN 0 bzw. SCREE 1. So müssen Sie auch bei der Textausgabe auf dem Grafikbildschirm vorsichtig sein, wenn Sie die untere Bildschirmzeile erreicht haben: Hier wird nicht hochgescrollt, sondern von oben links angefangen neu zu überschreiben. Es gibt zwei Möglichkeiten, wie Sie den Bildschirm löschen können:

1) vollständiges Löschen durch CLS

2) teilweise Löschung durch Überschreibung bzw. Neuschreiben des Textes mit der Hintergrundfarbe (ähnlich wie man mit weißem Tipp-Ex einen Fehler auf Papier übermalt bzw. mit einem Tintenkiller entfernt).

Den Effekt des Überschreibens mit der Hintergrundfarbe wollen wir zum Abschluß dieses Kapitels noch gerne zeigen:

siehe Kurzlisting OPEN 10 im Anhang

Die Überschreibung muß zu 100 Prozent identisch mit dem vorherigen Druckbild sein, denn sonst bleiben Teile übrig! Um dies zu ändern Sie Zeile 50 bitte folgendermaßen:

```
90 PRINT #1, "MSX-Cantuter"
```

Beim Starten des geänderten Programms mit RUN bleiben ein paar Reste des Wortes "Computer" auf dem Bildschirm erhalten, da das Wort "Cantuter" (Zeile 90) nicht 100 prozentig deckungsgleich mit "Computer" (Zeile 60) ist.

OPEN 1

```
         -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN Ø
30 MAXFILES=2
40 OPEN "CRT:" FOR OUTPUT AS #1
50 OPEN "LPT:" FOR OUTPUT AS #2
60 FOR N=1 TO 2
70 PRINT #N,"MSX-Computer sind SUPER"
80 NEXT N
90 CLOSE
```

OPEN 2

```
         -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN Ø
30 FOR N=1 TO 20
40 PRINT "Tolles MSX",
50 NEXT N
```

OPEN 3

```
         -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN Ø
30 OPEN "CRT:" FOR OUTPUT AS #1
40 FOR N=1 TO 20
50 PRINT #1,"Tolles MSX",
60 NEXT N
70 CLOSE
```

OPEN 4

```
         -*-*-*-*-*-
10 FOR N=1 TO 20
20 LPRINT "Tolles MSX",
30 NEXT N
```

OPEN 5

```
         -*-*-*-*-*-
10 OPEN "LPT:" FOR OUTPUT AS #1
20 FOR N=1 TO 20
30 PRINT #1,"MSX ist toll",
40 NEXT N
50 CLOSE
```

OPEN 6

```
         -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 PRINT "MSX in Grafik"
40 GOTO 40
```

OPEN 7

```
         -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 PRINT #1,"MSX in Grafik"
50 GOTO 50
```

OPEN 8

```
            -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 FOR N=1 TO 100
50 PRINT #1,"MSX ";
60 NEXT N
70 CLOSE
80 GOTO 80
```

OPEN 9

```
            -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 FOR N=1 TO 255 STEP 3
50 M=SIN(N/20)*50+100
60 DRAW "B M =n;,=m;"
70 PRINT #1,"MSX"
80 NEXT N
90 CLOSE
100 GOTO 100
```

OPEN 10

```
            -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 DRAW "B M 128,96"
50 COLOR 1
60 PRINT #1,"MSX-Computer"
70 DRAW "B M 128,96"
80 COLOR 15
90 PRINT #1,"MSX-Computer"
100 GOTO 40
```

Befehle   Grafik   VDP

Schreiben in bzw. Lesen aus den 9 Registern des ViDeoProzessors

siehe hierzu auch die Programme im Anhang: Programm 2
Programm 3
Programm 5
Programm 7
Programm 14

Bei diesem Befehl müssen wir die Zuweisung eines Wertes in binärer Form vornehmen. Deshalb sei an dieser Stelle noch einmal auf das dazu entsprechende Vorgehen hingewiesen:

Ein Byte besteht aus acht Bit, die jeweils nur den Wert 1 oder den Wert 0 annehmen können. Mit Hilfe der Zweierpotenz läßt sich z.B. aus acht Einsen die Dezimalzahl 255 errechnen. Sie weisen einer Variablen einen Binärwert zu, indem Sie vor die Zahl die Zeichenkombination '&B' setzen.

Also sieht die Eingabe von binär 00100100 an Variable A so aus:

```
A = &B00100100
```

Der Befehl VDP stellt eine Art von Variablenzuweisung dar. Um welche Variable es sich dabei handelt, zeigt uns die Aufschlüsselung der Buchstabenkombination VDP: Video Display Processor.

Mit dem Befehl VDP und einer in Klammern dahinter gesetzten Ziffer zwischen 0 und 8 können wir somit den VDP direkt ansprechen und mit Werten füttern bzw. Werte in den Registern abfragen. Die Werteingabe ist so lange von Ihnen mit Vorsicht zu genießen, bis Sie jedes Register des VDP sowie den allgemeinen Video-RAM-Aufbau genau kennengelernt haben, denn wir steigen mit der Behandlung des VDP-Befehls (hoffentlich allgemeinverständlich genug) in die Tiefen und Geheimnisse Ihres MSX-Computers ein, die neben Begeisterung auch so manche böse Überraschung in sich bergen können; jedoch kann im schlimmsten Fall durch falsches Programmieren nur Ihr Programm verloren gehen bzw. Sie können es nicht mehr auf dem Bildschirm sehen. Nie wird Ihr Computer durchs Programmieren Schaden nehmen!

Nichtsdestotrotz wollen wir uns die Register 0 bis 8 des VDP mal etwas genauer ansehen:

Der Inhalt der Register 0 bis 7 des VDP ist lesbar (PRINT VDP(N)) und beschreibbar (VDP(N)=M), während Register 8, das sogenannte Status Register, nur lesbar aber nicht beschreibbar ist.

Um die Register 0 bis 7 was ihren aktuellen Inhalt betrifft noch auf eine andere Art und Weise lesen zu können, sind alle diese Inhalte auch noch in den Speicherstellen &HF3DF bis &HF3E6 abgelegt. Der Inhalt von Register 8 ist ebenfalls noch einmal im RAM abgelegt und zwar an der Speicherstelle &HF3E7.

Der genauen Besprechung der einzelnen Register stellen wir jeweils den Registerinhalt in Binärform voran, wie er uns nach dem Einschalten des MSX-Computers (SCREEN 0, weiß als Vordergrund, Dunkelblau als Hintergrund) erscheint.

Register 0

aktueller Inhalt abrufbar durch:
PRINT PEEK(&HF3DF)
in Binärform (volle 8 Bit): 00000000
Zuweisung durch: VDP(0)=...

Lediglich Bit 0 und Bit 1 tragen bei diesem Register einen registrierten Inhalt, während Bit 2 bis Bit 7 ohne Funktion beliebige Inhalte enthalten dürfen.

Bit 0 ist so lange auf 0 zu setzen, bis wir parallel zum derzeit in Ihrem MSX-Computer sitzenden VDP einen zweiten VDP laufen lassen können. Erst zu diesem Zeitpunkt könnte man Bit 0 auf 1 setzen und somit den zweiten VDP einschalten. Sie sehen also: An den Ausbau des jetzigen MSX-Standards wurde bereits bei der Erstellung des derzeit gültigen Konzepts gedacht (wobei dann selbstverständlich eine Kompatibilität gewährleistet sein wird).

Bit 1 arbeitet mit Register 1 (Bit 3 und Bit 4) zusammen und bezieht sich auf den zur Zeit angewählten SCREEN-Mode. Bei den Bits 3 und 4 von Register 1 werden wir darauf näher eingehen, hier nur soviel: Bit 1 von Register 0 ist normalerweise = 0. Nur bei der Anwahl der hochauflösenden Grafik (SCREEN 2) ist hier eine 1 zu setzen.

Beispiele zur Verwendung von Register 0:

VDP(0)=&B00000001

Falls vorhanden wird der zweite VDP eingeschaltet

VDP(0)=&B00000010

Falls im Register 1 die Bits 3 und 4 den Wert 0 annehmen, schaltet Ihr MSX-Computer automatisch auf die hochaulösende Grafik = SCREEN 2 um.

Register 1

Aktueller Inhalt abrufbar durch:
PRINT PEEK(&HF3E0)
in Binärform (volle 8 Bit): 11110000
Zuweisung durch: VDP(1)=...

Bit 0 hat den Wert 0, wenn wir uns dazu entschlossen haben, normal große Sprites auf dem Bildschirm anzuwählen d.h. 8*8-Sprites sind auch wahrlich 8*8 Pixel groß, während 16*16-Sprites auch auf dem Bildschirm aus 16*16 Grafikpunkten zusammengesetzt sind.

Bit 0 hat den Wert 1, wenn wir mit vergrößerten Sprites arbeiten wollen. Dabei sind 8*8-Sprites dann auf dem Bildschirm 16*16 Punkte groß, denn jeder Punkt wird in einer Größe von 2*2 Punkten wiedergegeben. Das gleiche trifft für die 16*16-Sprites zu, die zu dem Zeitpunkt, wenn Bit 0 von Register 1 den Wert 1 einnimmt, vergrößert in einer Matrix von 32*32 Punkten auf dem Bildschirm dargestellt werden.

Übrigens: Wir können das Umsetzen von Bit 0 in Register 1 im MSX-BASIC dadurch ausdrücken, indem wir einfach den zweiten SCREEN-Parameter entsprechend setzen:

SCREEN ,0: 8*8-Sprites, normale Größe
SCREEN ,1: 8*8-Sprites, vergrößert
SCREEN ,2: 16*16-Sprites, normale Größe
SCREEN ,3: 16*16-Sprites, vergrößert

Sicherlich werden Sie nun fragen: Warum sollen wir das VDP-Register 1 und dort das erste Bit setzen oder nicht setzen, wenn dies doch viel einfacher mit dem SCREEN-Befehl und seinem zweiten Parameter klappt? Wie ich bereits zu Beginn dieses Kapitels darzuf hinwies: Wir steigen bei der Arbeit mit den VDP-Registern ziemlich tief in Ihren MSX-Computer ein. Diese Kenntnis ist für Sie als BASIC-Programmierer nicht in jedem Fall nützlich und sinnvoll. Aber denken Sie bitte an die Maschinensprachprogrammierer, die den SCREEN-Befehl gar nicht anwenden können, sondern viel mehr mit einem VDP-Register und INP- und OUT-Befehlen arbeiten können.

Bit 1 hat den Wert 0, wenn wir mit 8*8-Sprites arbeiten wollen. Hierbei handelt es sich vergleichsweise zu den SCREEN-Befehlen um die SCREEN ,0-Anwahl, jedoch wäre es auch möglich, daß wir mit SCREEN ,1 arbeiten, vorausgesetzt, Bit 0 von Register 1 wäre auf Vergrößerung = den Wert 1 eingestellt.

Bit 1 hat den Wert 1, wenn wir mit 16*16-Sprites arbeiten wollen. Hierbei handelt es sich vergleichsweise zu den SCREEN-Befehlen um die SCREEN ,2-Anwahl, jedoch wäre es auch möglich, daß wir mit SCREEN ,3 arbeiten, vorausgesetzt, Bit 0 von Register 1 wäre auf Vergrößerung = den Wert 1 eingestellt.

Die direkte Ansprache der Bits 0 und 1 von Register 1 des VDP unterscheidet sich jedoch im Vergleich zum SCREEN ,N-Kommando in einem wesentlichen Punkt: Während durch SCREEN ,N die vorher definierten Spritemuster im VRAM gelöscht werden, ist das bei Gebrauch des VDP-Befehls nicht der Fall.

Wir haben zur Demonstration dieses Zweckes ein Programm in diesem Buch abgedruckt, das Ihnen den direkten Übergang der vier verschiedenen Spritegrößen bzw. Spriteauflösungen deutlich macht:

siehe Programm 7 im Anhang

Bit 2 von Register 1 erfüllt zur Zeit noch keinen Nutzen. Es ist egal, ob Sie hier 0 oder 1 einsetzen; das Bildschirmaussehen verändert sich in keiner Weise.

Bit 3 und Bit 4 arbeiten eng miteinander zusammen und nicht nur das: Sie beziehen noch dazu die Meldung in ihr Tätigsein mit ein, wie Bit 1 in Register 0 aussieht.

Aus diesen drei Bits errechnet sich der gewünschte/bzw. der zur Zeit vorherrschende SCREEN. Um die 3 Bits genauer zu verstehen, müssen wir uns auf die folgende Tabelle beziehen:

Register 0 Bit 1	Register 1 Bit 3	Register 1 Bit 4	SCREEN
0	0	1	0
0	0	0	1
1	0	0	2
0	1	0	3

Sie können also mit Hilfe dieser 3 Bits den SCREEN nach Belieben wechseln, wobei auch dieses Vorgehen vor allem den Maschinensprachprogrammierern unter Ihnen einleuchten sollte, denn: Mit der Programmierung dieser 3 Bits können Sie den ersten Parameter im SCREEN-Befehl vergessen und brauchen nur noch mit diesen 3 Registern zu arbeiten. In Kombination mit den Bits 0 und 1 von Register 1 ist es uns gar möglich, auf den zweiten SCREEN-Parameter durch direkte Ansprache des VDP zu verzichten. Aber auch unter BASIC werden Sie die direkte Ansprache der Register und Bits des VDP noch zu schätzen lernen: Dann nämlich, wenn Sie den Aufbau des VRAM nach eigenen Wünschen umzugestalten gedenken. Mehr dazu bei der Beschreibung der Register 2 bis 6.

Bit 5 schaltet die automatische Maschineninterruptabfrage aus, wenn es auf 0 gesetzt ist und an, wenn es auf 1 gesetzt ist.

Erinnern Sie sich noch an die Kommandos ON SPRITE GOSUB und SPRITE ON/OFF/STOP? Genau darauf bezieht sich Bit 5 im VDP-Register 1. Sollte die Kollisionskontrolle von Sprites in Ihrem MSX-Computer stattfinden, haben Sie bisher den Befehl SPRITE ON verwendet. Daraufhin sprang Ihr MSX-Computer in ein vorher mit dem ON SPRITE GOSUB-Kommando gekennzeichnetes Unterprogramm, wo z.B. auf dem Bldschirm an der entsprechenden Bildschirmposition eine Explosion aufleuchtete. Mit SPRITE OFF stellten Sie bisher diese Spritekollisionskontrolle wieder ab. Das gleiche Ergebnis können Sie mit Bit 5 im VDP-Register 1 erreichen: Ist dieses Bit=1, entspricht die Funktion dem BASIC-Kommando SPRITE ON. Ist dieses Bit=0, entspricht die Funktion dem BASIC-Kommando SPRITE OFF. Hingegen läßt sich der MSX-Befehl SPRITE STOP nicht mit Hilfe eines VDP-Registers nachstellen.

Bit 6 hat den Wert 1, wenn wir ganz normal auf dem Bildschirm mit Vorder-, Hintergrund- und eventuell Umrandungsfarbe arbeiten wollen.

Setzen wir Bit 6 allerdings auf den Wert 0, so wird die Vordergrundfarbe auf dem Bildschirm = der Umrandungsfarbe gesetzt. Dies hat folgenden Sinn: Sie können nun auf dem Bildschirm ein Schrift- oder Grafikbild entwerfen und dies erst nach seiner Vollendung anschalten, indem Sie Bit 6 von Register 1 wieder mit der Zahl 1 belegen.

Bit 7 sollte immer den Wert 1 haben, da Ihr MSX-Computer über 16 Kilobyte VRAM verfügt. Ist Bit 7 von Register 1 auf 1 gesetzt, weiß damit der VDP darüber Bescheid, daß ihm zumindest 8 Kilobyte VRAM, im Normalfall sogar 16 Kilobyte VRAM zugeordnet sind. Bei früheren MSX-Versionen in der Erprobung hat man anscheinend in Erwägung gezogen, daß MSX-Computer auch nur über 4 Kilobyte VRAM verfügen könnten. In diesem Sinne wäre dann Bit 7 von Register 1 mit dem Wert 0 zur Kennzeichnung belegt worden.

Beispiele zur Verwendung von Register 1:

VDP(1)=&B11100011

Bit 0 = 1: vergrößerte Sprites (jeder Spritepunkt = 2*2 Bildschirmpunkte)

Bit 1 = 1: die großen Sprites (= 16*16-Sprites) werden dargestellt

Bit 0 und Bit 1 entsprechen somit dem BASIC-Kommando SCREEN ,3

Bit 2 = 0: keine Bedeutung

Bit 3 = 0 und Bit 4 = 0: ist gleichzeitig Bit 1 von Register 0 = 1, haben wir mit diesen 2 Bits SCREEN 2 (hochauflösende Grafik) erzeugt; ist gleichzeitig Bit 1 von Register 0 = 0, haben wir mit diesen 2 Bits SCREEN 1 angestellt

Bit 5 = 1: Interruptsteuerung für Spritekollision eingestellt; entspricht dem BASIC-Kommando SPRITE ON

Bit 6 = 1: normale Bildschirmdarstellung = die Grafik wird mit der Vordergrundfarbe sichtbar dargestellt

Bit 7 = 1: wir haben einen MSX-Computer mit 8 oder 16 Kilobyte VRAM

VDP(1)=&B10001000

Bit 0 = 0: Sprites werden in Originlagröße dargestellt

Bit 1 = 0: die kleinen Sprites (= 8*8) werden dargestellt

Bit 0 und 1 entsprechen somit dem BASIC-Kommando SCREEN ,0

Bit 2 = 0: keine Bedeutung

Bit 3 = 1 und Bit 4 = 0: Bit 1 von Register 0 muß auf 0 gesetzt sein. Wir befinden uns somit auf SCREEN 3 (Blockgrafik)

Bit 5 = 0: Spritekollisionskontrolle ausgestellt (entspricht dem BASIC-Kommando SPRITE OFF)

Bit 6 = 0: die Grafik wird unsichtbar mit durchscheinender Umrandungsfarbe dargestellt; erst wenn wir Bit 6 von Register 1 wieder auf 1 setzen, können wir den dargestellten Text bzw. die dargestellte Grafik auch sehen

Bit 7 = 1: wir haben einen MSX-Computer mit 8 oder 16 Kilobyte VRAM

Register 2

Aktueller Inhalt abrufbar durch:
PRINT PEEK(&HF3E1)
in Binärform (volle 8 Bit): 00000000
Zuweisung durch: VDP(2)=...

Nur die Bits 0 bis 3 sind mit Zahlwerten belegt. Diese Bits sind allerdings so zu lesen bzw. zu interpretieren, als wären sie in Wirklichkeit Bit 10, Bit 11, Bit 12 und Bit 13. Wären also diese 4 Bits jeweils mit einer 1 belegt, würde sich daraus die Binärzahl

0011110000000000

ergeben, die dezimal umgerechnet 15360 bedeutet.

Der kleinste Wert wäre binär

0000000000000000

was soviel bedeutet wie auch dezimal = 0. Wir haben im Anhang alle nur möglichen Werte aufgeführt, die mit diesen 4 Bits angegeben werden können. Soviel bereits an dieser Stelle: Ist nur Bit 0 auf 1 gesetzt (entspricht in der Berechnung Bit 10), haben wir den Dezimalwert 1024 erzeugt.

Was geben wir mit den 3 Bits von Register 2 an? Hiermit wird festgelegt, wo sich der Start des 'Pattern Name table' befindet d.h. wo der Aufbau des Bildschirms (Zeichen für Zeichen bzw. Punkt für Punkt) im VRAM abgelegt ist.

Weisen wir Register 2 keinen anderen Wert zu, so gilt für die einzelnen SCREENs folgendes:

	Start	Länge
SCREEN 0:	0	960
SCREEN 1:	6144	768
SCREEN 2:	6144	768
SCREEN 3	2048	768

Sie können die Position des Pattern Name Table im VRAM durch eine Veränderung der Bits in Register 2 verschieben. So können Sie z.B. auf SCREEN 0 mehrere Bildschirme im VRAM abspeichern und schnell durchblättern.

siehe Programm 2 im Anhang

Beispiele zur Verwendung von Register 2:

VDP(2)=&B0100

Der Start des Pattern Name Table soll an folgender Adresse im VRAM liegen:

&B0001000000000000 = Dezimal 5120

VDP(2)=&B1111

Der Start des Pattern Name Table soll an folgender Adresse im VRAM liegen:

&B0011110000000000 = Dezimal 15360

Register 3

Aktueller Inhalt abrufbar durch:
PRINT PEEK(&HF3E2)
in Binärform (volle 8 Bit): 00000000
Zuweisung durch: VDP(3)=...

Obgleich hier alle 8 Bits mit Werten belegt werden können, entspricht das Ergebnis nicht dem tatsächlichen Resultat. Vielmehr wird der Wert der 8 Bits in einem 16er-Bit von Position 13 bis Position 6 interpretiert.

Waren also diese 8 Bits von Register 3 jeweils mit einer 1 belegt, würde sich daraus die Binärzahl

0011111111000000

ergeben, die dezimal umgerechnet 16320 bedeutet.

Der kleinste Wert wäre binär

0000000000000000

was soviel bedeutet wie auch dezimal = 0. Wir haben im Anhang alle nur möglichen Werte aufgeführt, die mit diesen 8 Bits angegeben werden können. Soviel bereits an dieser Stelle: Ist nur Bit 0 auf 1 gesetzt (entspricht in der Berechnung Bit 6), haben wir den Dezimalwert 64 erzeugt.

Was geben wir mit den 8 Bits von Register 3 an? Hiermit wird festgelegt, wo sich der Start des 'Colour Table' befindet d.h. wo die farbliche Information über die Grafik/den Text auf dem Bildschirm im VRAM abgelegt ist.

Weisen wir Register 3 keinen anderen Wert zu, so gilt für die einzelnen SCREENs folgendes:

	Start	Länge
SCREEN 0:	nicht vorhanden	
SCREEN 1:	8192	32
SCREEN 2:	8192	6144
SCREEN 3	0	2048

Sie können die Startposition des Colour Table im VRAM durch eine Veränderung der Bits in Register 3 verschieben. So können Sie z.B. auf SCREEN 1 mehrere unterschiedliche Informationen zu den jeweils 32 Farbbytes im VRAM abspeichern und schnell durchblättern.

Beispiele zur Verwendung von Register 3:

VDP(3)=&B01010101

Der Start des Colour Table soll an folgender Adresse im VRAM liegen:

&B0001010101000000 = Dezimal 5440

VDP(3)=&B11111111

Der Start des Colour Table soll an folgender Adresse im VRAM liegen:

&B0011111111000000 = Dezimal 16320

Register 4

Aktueller Inhalt abrufbar durch:
PRINT PEEK(&HF3E3)
in Binärform (volle 8 Bit): 00000001
Zuweisung durch: VDP(4)=...

Nur die Bits 0 bis 2 sind mit Zahlwerten belegt. Diese Bits sind allerdings so zu lesen bzw. zu interpretieren, als wären sie in Wirklichkeit Bit 11, Bit 12 und Bit 13. Wären also diese 3 Bits jeweils mit einer 1 belegt, würde sich daraus die Binärzahl

0011100000000000

ergeben, die dezimal umgerechnet 14336 bedeutet.

Der kleinste Wert wäre binär

0000000000000000

was soviel bedeutet wie auch dezimal = 0. Wir haben im Anhang alle nur möglichen Werte angegeben, die mit diesen 3 Bits angegeben werden können. Soviel bereits an dieser Stelle: Ist nur Bit 0 auf 1 gesetzt (entspricht in der Berechnung Bit 11), haben wir den Dezimalwert 2048 erzeugt.

Was geben wir mit den 3 Bits von Register 4 an? Hiermit wird festgelegt, wo sich der Start des 'Pattern Generator Table' befindet d.h. das Aussehen der Zeichen bzw. die hochauflösende Grafik.

Weisen wir Register 4 keinen anderen Wert zu, so gilt für die einzelnen SCREENs folgendes:

	Start	Länge
SCREEN 0:	2048	2048
SCREEN 1:	0	2048
SCREEN 2:	0	6144
SCREEN 3	nicht vorhanden	

Sie können die Startposition des Pattern Generator Table im VRAM durch eine Veränderung der Bits in Register 4 verschieben. So können Sie z.B. auf SCREEN 0 mehrere verschiedene Zeichensätze im VRAM abspeichern und je nach Belieben den gewünschten Zeichensatz auswählen.

siehe Programm 3 und 14 im Anhang

Beispiele zur Verwendung von Register 4:

VDP(4)=&B010

Der Start des Pattern Generator Table soll an folgender Adresse im VRAM liegen:

&B0001000000000000 = Dezimal 4096

VDP(4)=&B111

Der Start des Pattern Generator Table soll an folgender Adresse im VRAM liegen:

&B0011100000000000 = Dezimal 14336

Register 5

Aktueller Inhalt abrufbar durch:
PRINT PEEK(&HF3E4)
in Binärform (volle 8 Bit): 00000000
Zuweisung durch: VDP(5)=...

Nur die Bits 0 bis 6 sind mit Zahlwerten belegt. Diese Bits sind allerdings so zu lesen bzw. zu interpretieren, als wären sie in Wirklichkeit Bit 7 bis Bit 13. Wären also diese 7 Bits jeweils mit einer 1 belegt, würde sich daraus die Binärzahl

0011111110000000

ergeben, die dezimal umgerechnet 16256 bedeutet.

Der kleinste Wert wäre binär

0000000000000000

was soviel bedeutet wie auch dezimal = 0. Wir haben im Anhang alle nur möglichen Werte angegeben, die mit diesen 7 Bits angegeben werden können. Soviel bereits an dieser Stelle: Ist nur Bit 0 auf 1 gesetzt (entspricht in der Berechnung Bit 7), haben wir den Dezimalwert 128 erzeugt.

Was geben wir mit den 7 Bits von Register 5 an? Hiermit wird festgelegt, wo sich der Start des 'Sprite Attribute Table' befindet d.h. wo sich zur Zeit Sprites auf dem Bildschirm befinden (X,Y-Koordinatenangabe), welche Farbe sie haben und auf welches Spritemuster sie sich im VRAM beziehen.

Weisen wir Register 5 keinen anderen Wert zu, so gilt für die einzelnen SCREENs folgendes:

	Start	Länge
SCREEN 0:	nicht vorhanden	
SCREEN 1:	6912	128
SCREEN 2:	6912	128
SCREEN 3	6912	128

Sie können die Startposition des Sprite Attribute Table im VRAM durch eine Veränderung der Bits in Register 5 verschieben. So können Sie z.B. auf SCREEN 1 mehrere Spritepositionen und Spritefarben im VRAM abspeichern und schnell verändern.

Beispiele zur Verwendung von Register 5:

VDP(5)=&B0101010

Der Start des Sprite Attribute Table soll an folgender Adresse im VRAM liegen:

&B0001010100000000 = Dezimal 5376

VDP(5)=&B1111111

Der Start des Sprite Attribute Table soll an folgender Adresse im VRAM liegen:

&B0011111110000000 = Dezimal 16256

Register 6

Aktueller Inhalt abrufbar durch:
PRINT PEEK(&HF3E5)
in Binärform (volle 8 Bit):00000000
Zuweisung durch: VDP(6)=...

Nur die Bits 0 bis 2 sind mit Zahlwerten belegt. Diese Bits sind allerdings so zu lesen bzw. zu interpretieren, als wären sie in Wirklichkeit Bit 11, Bit 12 und Bit 13. Wären also diese 3 Bits jeweils mit einer 1 belegt, würde sich daraus die Binärzahl

0011100000000000

ergeben, die dezimal umgerechnet 14336 bedeutet.

Der kleinste Wert wäre binär

0000000000000000

was soviel bedeutet wie auch dezimal = 0. Wir haben im Anhang alle nur möglichen Werte angegeben, die mit diesen 3 Bits angegeben werden können. Soviel bereits an dieser Stelle: Ist nur Bit 0 auf 1 gesetzt (entspricht in der Berechnung Bit 11), haben wir den Dezimalwert 1024 erzeugt.

Was geben wir mit den 3 Bits von Register 6 an? Hiermit wird festgelegt, wo sich der Start des 'Sprite Generator Table' befindet d.h. wo je nach SCREEN-Kommando das Aussehen von bis zu 256 Sprites der Größe 8*8 bzw. von bis zu 64 Sprites der Größe 16*16 im VRAM abgelegt ist.

Weisen wir Register 6 keinen anderen Wert zu, so gilt für die einzelnen SCREENs folgendes:

	Start	Länge
SCREEN 0:	nicht vorhanden	
SCREEN 1:	14336	2048
SCREEN 2:	14336	2048
SCREEN 3	14336	2048

Sie können die Startposition des Sprite Generator Table im VRAM durch eine Veränderung der Bits in Register 6 verschieben. So können Sie z.B. auf SCREEN 1 mehrere Gruppen von jeweils bis zu 256 8*8-Sprites im VRAM abspeichern und schnell durchblättern.

Beispiele zur Verwendung von Register 6:

VDP(6)=&B010

Der Start des Sprite Generator Table soll an folgender Adresse im VRAM liegen:

&B0001000000000000 = Dezimal 4096

VDP(6)=&B111

Der Start des Sprite Generator Table soll an folgender Adresse im VRAM liegen:

&B0011100000000000 = Dezimal 14336

Register 7

Aktueller Inhalt abrufbar durch:
PRINT PEEK(&HF3E6)
in Binärform (volle 8 Bit): 11110100
Zuweisung durch: VDP(7)=...

Je nach angewähltem SCREEN haben alle 8 Bits (Bit 0 bis Bit 7) eine unterschiedliche Bedeutung:

Arbeiten wir auf SCREEN 0, so tragen die Bits 0 bis 3 die Information über die derzeit aktuelle Hintergrundfarbe, während die Bits 4 bis 7 die Information über die Vordergrundfarbe enthalten. Wie bekannt sein dürfte, gibt es bei SCREEN 0 als diesbezüglich einzigem SCREEN nur 2 Farben, die als Hinter- und Vordergrundfarbe aus den 15 zur Verfügung stehenden Farbtönen ausgewählt werden können. Man kann diese Farbinformation am einfachsten in das Register 7 eingeben, indem man sich die 8 Bits als 2 Hexadezimalwerte umrechnet, denn: bei 2 Hexadezimalziffern belegt jede genau 4 Bits, so daß man nebeneinander die Vorder- und Hintergrundfarbe deutlich sehen kann (z.B. F1: Vordergrundfarbe=F=15=weiß, Hintergrundfarbe=1=schwarz).

Arbeiten wir auf SCREEN 1, 2 oder 3 tragen die Bits 0 bis 3 von Register 7 die Informationen über die Umrandungsfarbe, während die Bits 4 bis 7 wiederum für die Vordergrundfarbe verantwortlich sind. Auch hier erscheint es ratsam, Register 7 direkt mit Hexadezimalwerten zu speisen, denn gleich wie bei SCREEN 0 reicht hierzu eine Hexadezimalzahl aus.

Beispiele zur Verwendung von Register 7:

VDP(7)=&H11110001 = hexadezimal F1

Falls wir uns auf SCREEN 0 befinden, ist die Vordergrundfarbe=F=15=weiß und die Hintergrundfarbe=1=schwarz. Befinden wir uns auf SCREEN 1, 2 oder 3 ist die Vordergrundfarbe ebenfalls F=15=weiß, während die Umrandungsfarbe 1=schwarz ist.

Register 8

Wie bereits vorhin darauf hingewiesen, ist dieses Register nicht zum Beschreiben, sondern im Gegensatz zu den Registern 0 bis 7 nur zum Lesen zu gebrauchen.

Aktueller Inhalt abrufbar durch:
PRINT PEEK(&HF3E7)
in Binärform (volle 8 Bits): 11001101
lesbar durch: PRINT VDP(8)

Bit 0 bis Bit 4: Wie Sie wissen, gibt es bei der Darstellung Schwierigkeiten, wenn sich mehr als 4 Sprites auf einer Bildschirmhorizontalen begegnen: das fünfte Sprite bzw. weitere hinzukommende Sprites werden nicht mehr vollkommen dargestellt bzw. es gilt das Gesetz: die Sprites mit den niedrigsten Nummern d.h. sie liegen dreidimensional gesehen oben, bleiben erhalten, während die überzähligen Sprites mit höheren Nummern nicht mehr dargestellt werden (eventuell sind selbst bei 8 Sprites noch Teile sichtbar).

Bit 0 bis 4 enthält die Information über die Nummer des fünften Sprites (falls es soweit kommen sollte), das sich zu 4 bereits vorhandenen Sprites auf einer Bildschirmhorizontalen hinzugesellt und die störende Überlagerung herbeiführt.

siehe Programm 5 im Anhang

Bit 5 hat den Wert 1, wenn 2 Sprites auf dem Bildschirm zusammenstoßen und die Interruptsteuerung aktiviert wurde (indem Bit 5 von Register 1 auf 1 gesetzt wurde oder die BASIC-Befehle ON SPRITE GOSUB und SPRITE ON gegeben wurden). Stoßen keine Sprites zusammen, ist der Bitwert = 0. Jedoch müssen Sie vorsichtig sein, wenn Sie das VDP-Register 8 abfragen, um zu erfahren, ob 2 Sprites zusammengestoßen sind, denn durch die VDP-Abfrage von Register 8 wird die Interruptsteuerung ausgeschaltet (entspricht dem BASIC-Kommando SPRITE OFF). Besser ist es, zu diesem Zweck, die Speicherstelle &HF3E7 abzufragen, die ständig entsprechend dem Inhalt des VDP-Registers aktualisiert wird.

Bit 6 hat den Wert 1, wenn sich mehr als 4 Sprites auf einer Bildschirmzeile befinden. Hiernach ist es ratsam, die Nummer des Verursacher-Sprites durch die Bits 0 bis 4 des Registers 8 zu entschlüsseln, um schnellstmöglich gegen diesen unschönen Effekt Gegenmaßnahmen einleiten zu können. Bit 6 hat den Wert 0, wenn zur Zeit nicht mehr als 4 Sprites auf jeder Horizontalen (punktbezogen) zu finden sind.

Bit 7 schließlich arbeitet indirekt mit sämtlichen VDP-Kommandos zusammen: Dieses Bit wir 50 mal pro Sekunde auf den Wert 1 gesetzt und zwar jedes mal dann, wenn ein Bildschirmaufbau beendet ist. Für BASIC-Programmierer wird dies wegen der Langsamkeit der Programmiersprache BASIC andauernd zutreffen und somit nicht von Bedeutung sein. Erst unter Maschinensprache kann dieses Bit wichtige Aufschlüsse liefern.

Beispiel zur Verwendung von Register 8:

PRINT BIN$(VDP(8))
unter BASIC gleichbedeutend mit: PRINT BIN$(PEEK(&HF3E7))
Ergebnis: 11111111

Bit 0 bis Bit 4: es gibt zur Zeit ein Sprite (Nummer binär 11111=dezimal 31), das zu 4 bereits vorhandenen anderen Sprites in einer Horizontalen hinzugestoßen ist

Bit 5: zudem besagt diese 1, daß 2 Sprites auf dem Bildschirm miteinander zusammengestoßen sind

Bit 6: diese 1 bestätigt die getroffene Aussage für die Bits 0 bis 4: mehr als 4 Sprites befinden sich auf einer Horizontalen (punktbezogen)

Bit 7: der Bildschirm wurde wie jeweils alle 1/50 Sekunde wieder einmal neu aufgefrischt, darum ist dieses Bit = 1.

Befehle Grafik BASE

Dimensionierte Variable, die über den VRAM-Aufbau informiert

siehe hierzu auch die Programme im Anhang: BASE 1
BASE 2
BASE 3
BASE 4
BASE 5
BASE 6
BASE 7
BASE 8
BASE 9
BASE 10
BASE 11
BASE 12
BASE 13
BASE 14
BASE 15
Programm 14

Um diesen Befehl richtig nutzen zu können, müssen Sie bereits einiges an Wissen über die Video-RAM-Abspeicherung im MSX-System mitbringen. Der Befehl BASE erstellt zwar direkt keine Textgrafik, dafür weist er Sie aber genau darauf hin, wie der Video-RAM-Speicher im entsprechenden Screen gerade aufgebaut ist. Außerdem können Sie den Befehl BASE dazu verwenden, den Video-RAM-Speicher nach eigenem Belieben zu gestalten.

Genaugenommen handelt es sich bei BASE mehr um eine Variable als um einen Befehl. Je nachdem, welchen Wert Sie BASE zuweisen, wird der Aufbau des VRAM direkt oder indirekt davon betroffen.

Fangen wir an, den Befehl BASE (oder besser gesagt: die Variable BASE) mit seinen verschiedenen Inhalten zu untersuchen. Die BASE-Variable besteht aus einem dimensionierten Feld von 20 Werten, das bereits beim Einschalten des MSX-Computers mit Zahlen belegt ist. Tippen Sie folgendes Programm ein, um sich davon zu überzeugen:

siehe Kurzlisting BASE 1 im Anhang

Ergebnis:		
	0 = 0	1 = 0
	2 = 2048	3 = 0
	4 = 0	5 =6144
	6 = 8192	7 = 0
	8 = 6912	9 = 14336
	10 = 6144	11 = 8192
	12 = 0	13 = 6912
	14 = 14336	15 = 2048
	16 = 0	17 = 0
	18 = 6912	19 = 14336

Interessant, aber was soll das? Gehen wir der Reihe nach vor: Die 20 abgebildeten Zahlwerte sind nicht gemeinsam zu gebrauchen, sondern sie stellen vielmehr vier Gruppen zu je fünf Zahlen dar. Diese vier Gruppen zu je fünf Zahlen sind den verschiedenen Bildschirmdarstellungen zugeordnet, die Sie mit dem MSX-System und dem eingebauten GI-Grafikchip ansteuern können:

Werte 0 bis 4: SCREEN 0
Werte 5 bis 9: SCREEN 1
Werte 10 bis 14: SCREEN 2
Werte 15 bis 19: SCREEN 3

Gehen wir noch weiter in die Details: Die fünf Zahlenwerte beziehen sich auf jeweils fünf verschiedene Anfangsadressen im Video-RAM-Speicher, als da sind:

Wert 0: hier ist der Bildschirmaufbau abgespeichert, bezogen auf jeweils 256 verschiedene Zeichen
Wert 1: hier ist die Farbe der dargestellten Zeichen abgespeichert (nicht in SCREEN 0 und 3)
Wert 2: hier ist das Aussehen der einzelnen Charakterzeichen bzw. die Grafik Punkt für Punkt abgespeichert
Wert 3: hier ist das abgespeichert, was mit der Farbe und der Position der Sprites zu tun hat (nicht in SCREEN 0)
Wert 4: hier befindet sich der Speicherbereich, in dem das Spriteaussehen zu finden ist (nicht in SCREEN 0)

Die Werte 5 bis 9 enthalten die entsprechenden Informationen für SCREEN 1, die Werte 10 bis 14 die Informationen für SCREEN 2 und letztendlich die Werte 15 bis 19 die Informationen für SCREEN 3.

Wie aus der obigen Aufstellung leicht zu erkennen ist, kann man in SCREEN 0 und SCREEN 3 nur Teile des Video-RAM laut BASE-Liste sinnvoll verwenden. Führen wir deshalb die Beispiele in SCREEN 1 durch:

1) Die Anfangsadresse für unseren Bildschirminhalt liegt laut BASE-Tabelle bei Adresse 6144. Um das zu überprüfen, löschen wir den Bildschirm (mit SCREEN 1) und poken an Adresse 6144 die Zahl 2:

```
SCREEN 1
VPOKE 6144,2
```

Sie müssen achtgeben, daß Sie nicht aus Versehen den Befehl POKE verwenden. Hier arbeiten wir ja im Video-RAM-Speicher und der Befehl zur Veränderung dieser Speicherstellen lautet VPOKE (Video-POKE) und nicht POKE.

Was ist geschehen? Eigentlich fast nichts, aber bitte übersehen Sie nicht die Bildschirmecke ganz weit links oben. Es könnte gar sein, daß Sie diese Bildschirmecke gar nicht einsehen können. Dann müssen Sie Ihren Fernsehapparat oder Monitor so justieren, bis Sie auch in die äußerst linke Bildschirmecke schauen können. Dort müßte ein Smiley-Gesicht abgebildet sein. Schauen wir in unserer ASCII-Tabelle nach (siehe Anhang), erkennen wir bei ASCII-Zeichen 2 dieses lachende Gesicht wieder. Wir haben unserem MSX-Computer also mit dem Befehl VPOKE 6144,2 zu verstehen gegeben, daß er in die Bildschirmecke links oben ein Smiley-Gesicht malen soll. Damit wir es alle sehen können, setzen wir das Smiley-Gesicht in die Mitte der ersten Bildschirmzeile. Im verwendeten SCREEN 1 haben wir 32 Zeichen pro Zeile, also wird zur Mittenpositionierung in Speicherstelle 6144 + 15 ein weiteres Smiley-Gesicht eingegeben:

```
VPOKE 6144 + 15,2
```

Von diesem einen Gesicht ausgehend, wollen wir nur eine ganze Spalte auf dem Bildschirm abbilden, von oben nach unten:

siehe Kurzlisting BASE 2 im Anhang

Wie Sie sehen, stört sich Ihr MSX-Computer gar nicht daran, ob Sie die Funktionstastenlatte auf dem Bildschirm erscheinen lassen (und damit vor dem Überschreiben schützen d.h. Sie haben nur noch 23 anstelle von 24 Arbeitszeilen), wenn Sie im VRAM-Speicher herumpoken. Auf einmal erscheint auch ein Smiley-Gesicht in Zeile 24 und das ist durch Eingabe eines weiteren VPOKEs wieder dort wegzubekommen, es sei denn, Sie definieren die Funktionstastenlatte neu oder löschen den Bildschirm mit CLS bzw. SCREEN 1 (Sie können das Smiley-Gesicht aus der Funktionstastenlatte auch verschwinden lassen, indem Sie CLS eingeben oder die SHIFT-Taste betätigen).

Wir können uns diesen zweiten Bildschirmrahmen (der eigentliche Rahmen wird ja mit Hilfe des dritten Parameters im COLOR-Befehl verändert) zunutze machen, indem wir mit WIDTH unser Bildschirmfeld kleiner halten als möglich und die Funktionstastenlatte beibehalten. Probieren Sie folgendes Beispiel aus und modifizieren Sie es nach eigenen Vorstellungen:

siehe Kurzlisting BASE 3 im Anhang

Links, rechts und unten ist Ihr Bildschirm jetzt mit Smiley-Gesichtern umgeben. So lange Sie kein CLS und auch keinen SCREEN-Befehl eingeben, bleibt dieser nach oben offene Bildschirmrahmen erhalten. Sie können den zweiten Bildschirmrahmen auch dadurch wieder löschen, indem Sie die Bildschirmbreite verändern oder aber erneut in die entsprechenden Bildschirmstellen poken.

Sie können dem Programm BASE 3 entnehmen, daß der Bildschirm in SCREEN 1 von Speicherstelle 6144 (BASE (5)) bis Speicherstelle 6911 reicht, das sind 768 Speicherplätze, denn 32 Zeichen pro Zeile * 24 (Anzahl der Zeilen) ergibt 768.

Wie sieht es im Vergleich dazu in SCREEN 0 aus? BASE (0) teilt uns den Bildschirmanfang mit:

PRINT BASE (0)

Ergebnis: 0

Wir haben hier 24 Zeilen zu jeweils 40 Zeichen. Also ist der Bildschirmspeicher bei SCREEN 0 960 Zeichen lang - von Adresse 0 bis Adresse 959.

Schauen wir uns noch die Informationen über die zwei Grafikbildschirme (SCREEN 2 und SCREEN 3) an, soweit es den Aufbau (nicht die Farbgebung und auch nicht die Sprites) anbetrifft:

Für SCREEN 2: PRINT BASE (10)

Ergebnis: 6144

Für SCREEN 3: PRINT BASE (15)

Ergebnis: 2048

Bevor wir uns nun mit der Farbe auf Grafik und Textbikldschirmen beschäftigen, hier die technischen Daten zum grobauflösenden Bildschirm SCREEN 3: Die Darstellung beginnt im Speicher (siehe Variable BASE (15)) bei Speicherstelle 2048 und ist ebenfalls 768 Speicherstellen lang.

Der Übersicht halber hier noch einmal alle vier Screens, soweit es die Bildschirmdarstellung betrifft, auf einen Blick:

SCREEN 0, Anfang 0, Länge 960, Variable BASE (0)
SCREEN 1, Anfang 6144, Länge 768, Variable BASE (5)
SCREEN 2, Anfang 6144, Länge 768, Variable BASE (10)
SCREEN 3, Anfang 2048, Länge 768, Variable BASE (15)

Schauen wir uns nun den zweiten Teil der BASE-Adressen an. Hier ist die Farbe der Buchstaben bzw. Grafiken abgespeichert:

2) Die Variable BASE(1) gibt uns den Wert 0 an. Das kann eigentlich nicht sein, werden Sie richtig erkannt haben, denn BASE(0) hatte bereits den Wert 0. Hier steht die 0 lediglich dehalb, weil wir in SCREEN 0 gar nicht die Farbe differenziert angeben können, es sei denn, wir verwenden den COLOR-Befehl. Dieser Befehl gilt im gleichen Augenblick für den gesamten Bildschirm, genauso wie in SCREEN 1 und braucht lediglich zwei Speicherplätze. Deshalb wird im VRAM für SCREEN 0 kein gesonderter Farbtish angelegt.

Wenden wir uns gleich SCREEN 1 zu:

```
10 SCREEN 1
20 PRINT BASE (6)
```

Ergebnis: 8192

Der 'Farbtisch' beginnt bei Speicheradresse 8192 im VRAM. Er ist allerdings nur 32 Zeichen lang und die sind geradezu auch noch irrsinnig für die Anwendung verteilt. Bevor wir uns näher damit auseinandersetzen, geben Sie folgendes Beispiel ein:

siehe Kurzlisting BASE 4 im Anhang

Wenn Sie dieses Programm laufen lassen, verkehrt sich der Bildschirm je nach Anzahl und Unterschiedlichkeit der dargestellten Zeichen in ein wirres, vielleicht gar unleserliches Etwas. Um zu sehen, warum dies so ist, setzen Sie die Farben wieder in den Urzustand zurück (durch Eingabe von SCREEN 1 - eine einfache Farbveränderung mit COLOR reicht hier nicht mehr allein aus) und geben folgende Programmzeilen zusätzlich ein:

siehe Kurzlisting BASE 4a im Anhang

Nun starten Sie das gesamte Programm mit RUN. Jeweils eine Folge von acht Zeichen (z.B. H bis O oder 0 bis 7) haben sich als Gruppe in Hintergrund- und Vordergrundfarbe geändert. Aus diesem Grund sehen wir beim Drücken der Taste 7 und der Taste 8 ganz andere Farben, als wenn wir die Tasten 6 und 7 drücken (die letztgenannten Zeilenzeichen gehören halt zur selben Gruppe, während Zahlzeichen 8 die Farbe einer anderen 8er-Gruppe angenommen hat.

Wie ist nun die Eingabe der 32 Bytes organisiert? Wir wollen dazu die Speicherstelle 8198 (sie enthält die Farbzuweisung für die Zahlzeichen 0 bis 7) verändern.

```
VPOKE 8198,255
```

Ergebnis: weißer Untergrund, weißes Zeichen

```
VPOKE 8198,0
```

Ergebnis: durchsichtiger Untergrund, durchsichtiges Zeichen

```
VPOKE 8198,31
```

Ergebnis: weißer Untergrund, schwarzes Zeichen

Um klarer zu sehen, schauen wir uns die eingegebenen Zahlwerte im Binärformat (nur mit 0 und 1) an:

```
255 = 11111111
  0 = 00000000
 31 = 00011111
```

Wir müssen uns die acht Ziffern in zwei Vierergruppen aufteilen. Dann hat die Dezimalzahl 255 die binäre Bedeutung 1111 und 1111, was wiederum ins dezimale System übersetzt heißt: 15 und 15. Die erste Zahl (die links stehenden vier Einsen) geben die Vordergrundfarbe an (Farbe 15 ist weiß), die zweite Zahl besagt, wie der Hintergrund des Zeichens beschaffen ist (in diesem Fall auch weiß). Bei unserem dritten Beispiel (31) bzw. binär 0001 1111) ist somit der Vordergrund (das eigentliche Zeichen) schwarz (binär 0001=dezimal 1), während der Hintergrund weiß (binär 1111=dezimal 15) auf dem Bildschirm erscheint.

Zum Verständnis noch zwei weitere Beispiele in umgekehrter Reihenfolge (dann können Sie mit dem Probieren beginnen):

Vordergrund Farbe 13, Hintergrund Farbe 7
binär 1101 0111; dezimal 215
VPOKE 8198,215

Vordergrund Farbe 7, Hintergrund Farbe 13
binär 0111 1101; dezimal 125
VPOKE 8198,125

Gehen wir weiter zur Farbgebung in SCREEN 2. Hier sagt uns der Wert von BASE (11), daß der Farbspeicher der hochauflösenden Grafik bei Speicherstelle 8192 beginnt. Im Farb-VRAM sind 6144 Zeichen gespeichert, was sich mit einer einfachen Rechenaufgabe leicht beweisen läßt: 192 Zeilen lassen sich in jeder Zeichengröße mit beliebigen Farben belegen; da es 32 Zeichen pro Zeile gibt, sagt uns die Multiplikation von 32 mit 192 das bereits bekannte Endresultat: 6144.

Schauen wir uns diese Farbenvielfalt doch einmal in einem Beispielprogramm etwas genauer an:

siehe Kurzlisting BASE 5 im Anhang

Ein wahrlich bunt schillernder Bildschirm, der uns in dieser Form den VRAM-Speicheraufbau zeigt.

Doch ehrlich gesagt sehen wir gar nicht alles, denn in jedem Farbfleck von 8*1 Punkten (wie sie uns als winzige waagerechte Balken auf dem Bildschirm erscheinen) sind jeweils zwei Farben, nämlich eine Vorder- und eine Hintergrundfarbe gespeichert - mehr nicht. Die Farbabspeicherung für Vorder- und Hintergrundfarbe in einem Zeichen geht genauso vor sich, wie dies bereits in SCREEN 1 beschrieben wurde: die ersten vier Bit für die Vordergrundfarbe, die letzten vier Bit für die Hintergrundfarbe. Hieraus lassen sich dann Hexadezimalzahlen zur leichteren Eingabe errechnen wie z.B. Vordergrundfarbe 7 und Hintergrundfarbe FF ergibt binär 0111 1111 oder dezimal 127.

Schließlich noch zur Farbgebung in SCREEN 3: Die grobauflösende Grafik ist in 64*48 Bildpunkte aufgeteilt. Wieder sind in einem Byte zwei Dinge abgespeichert: Die Vorder- und die Hintergrundfarbe. Also können wir erneut eine Rechnung aufstellen, wieviel Speicherplätze dieser Farbbildschirm verbraucht: (64/2)*48 = 1536 Zeichen.

Der Übersichtlichkeit halber hier noch einmal alle vier SCREENs, soweit es die Farbdarstellung allein betrifft, auf einen Blick:

SCREEN 0, Anfang /, Länge /, Variable BASE (1)
SCREEN 1, Anfang 8192, Länge 32, Variable BASE (6)
SCREEN 2, Anfang 8192, Länge 6144, Variable BASE (11)
SCREEN 3, Anfang 0, Länge 1536, Variable BASE (16)

Schauen wir und nun den dritten Teil der BASE-Adressen an. Hier ist das Aussehen der einzelnen Zeichen auf dem Bildschirm Punkt für Punkt vermerkt:

3) Da wir in SCREEN 0 keine Grafikdarstellung haben, wird diese Abspeicherung dem Aussehen der einzelnen Zeichen zugeordnet. Diese Abspeicherung beginnt bei Adresse 2048 und zieht sich über 2048 Speicherplätze hin. Wie und wo ist nun ein Zeichenaussehen abgespeichert?

Hierzu müssen wir uns daran erinnern, daß alle Zeichen vom Computer als Zahlen interpretiert werden; so wird das Leerzeichen für den Computer als CHR$(32) verstanden, die Anführungszeichen oben als CHR$(34) und schließlich der Buchstabe A als CHR$(65) (siehe auch unter der Befehlserklärung zu ASC und CHR$())

Schauen wir einmal nach, wie unser MSX-Computer die Buchstaben und Zeichen im Speicher abgelegt hat:

<u>siehe Kurzlisting BASE 6 im Anhang</u>

```
Ergebnis: 32
          80
          136
          136
          248
          136
          136
          0
```

Dies reicht aber zur Darstellung des Buchstabenzeichens (in unserem Fall des A) noch nicht aus, denn hinter jeder Dezimalzahl steckt für den Computer die ein/aus bzw. die 1/0-Verschlüsselung:

<u>siehe Kurzlisting BASE 7 im Anhang</u>

```
Ergebnis: 00100000  =    1
          01010000  =   1 1
          10001000  =  1    1
          10001000  =  1    1
          11111000  =  111111
          10001000  =  1    1
          10001000  =  1    1
          00000000  =
```

Sie werden sich sicherlich fragen, warum bei dieser 8*8-Matrix die rechten drei Spalten frei bzw. undefiniert geblieben sind. Dies hat zwei Gründe: Zum einen muß mindestens eine Spalte frei bleiben,um die Abgrenzung zum sich nach rechts anschließenden Zeichen zu gewährleisten. Die anderen zwei Spalten, die frei geblieben sind, haben mit der Eigenschaft der hochauflösenden Grafik der MSX Computer zu tun, denn es sollen ja schließlich bis zu 40 Zeichen pro Zeile dargestellt werden. Da auch der Textbildschirm SCREEN 0 mit dem Grafikprozessor erzeugt wird, schrumpft die Darstellung wegen der 40 Zeichen/Zeile auf eine Matrix von 6*8 zusammen, denn 6 * 40 = 240, während 8 * 40 =320 sind - wir haben allerdings 'nur' 256 Punkte pro Zeile zur Verfügung.

Außerdem muß noch die Frage beantwortet werden, warum die unterste Zeile (Zeile 7) beim Buchstaben A frei bleibt: Der MSX-Zeichensatz besteht, wie Sie wissen, aus Klein- und Großbuchstaben. Die kleinen Buchstaben wie j und g brauchen eine Unterlänge. Um dieselbige auch auf dem Bildschirm anzuzeigen, bleibt bei den meisten Zeichen die letzte Zeile zur Unterscheidung frei.

Sie können die Matrix der Buchstaben und Zeichen in ihrem Aussehen so verändern, wie es Ihnen beliebt. Um diesem Zweck Rechnung zu tragen, haben wir an anderer Stelle in diesem Buch einen sogenannten Charaktergenerator veröffentlicht:

siehe Programm 14 im Anhang

Also der Verwirklichung Ihrer Kreativität sind keine Grenzen gesetzt.

In SCREEN 1 läuft der Umgang mit dem im Speicher abgelegten Zeichenaussehen genauso ab, wie in SCREEN 0 bereits erklärt, nur ist hier die Startadresse im VRAM die 0. Die Länge dieses Speicherabschnittes beträgt ebenfalls 2048 Bytes.

Da in SCREEN 1 die Zeichen etwas auseinandergezogen sind (nicht mehr 40 wie im SCREEN 0, sondern nun 32 Zeichen/Zeile), wird hier nun auch das vollständige Zeichenaussehen zur Anzeige gebracht (Matrix 8*8, denn 8 * 32 ergibt 256, genau das ist auch die grafische Auflösung unserer MSX-Computer). Was läge da näher, als einen Zeichensatz extra für SCREEN 1 zu entwickeln. Dies haben wir auch getan und Sie können - wenn nicht bereits geschehen - gleich mit dem Eintippen des dargestellten Computerzeichensatzes mit Hilfe des Charactergenerators beginnen.

Nun noch zu den Grafikbildschirmen: Bei unseren vorherigen Grundüberlegungen, soweit sie sich bereits mit der Grafik beschäftigten, haben wir uns seinerzeit die Frage gestellt: Wie viele Speicherplätze braucht man, um 256 * 192 Punkte darzustellen. Bis jetzt wissen wir erst einmal, wo die Abspeicherung beginnt. Bei SCREEN 2 hatten wir vorhin zur Demonstration der Farbdarstellung ein kurzes Programm eingetippt. Da die hochauflösende Punktgrafik erst durch Farbe sichtbar wird, müssen wir in diesem Demonstrationsprogramm nun Farbe und hochauflösende Grafik erstellen. Erst dadurch wird auch eindeutig sichtbar, daß es in der Farbdefinition auch sowohl eine Vorder- als auch eine Hintergrundfarbe gibt. Geben Sie zu diesem Demonstrationszweck bitte folgendes Programm ein:

siehe Kurzlisting BASE 8 im Anhang

Erst wird in Zeile 20 nach und nach punktweise (in einem Byte sind jeweils acht Punkte definiert - denken Sie an Punkt gesetzt, Punkt nicht gesetzt oder 1, 0) ein Muster auf dem Bildschirm erzeugt und anschließend wird die Farbe darüber geschüttet (je acht Punkte in der Horizontalen werden 'nur' zwei verschiedene Farben akzeptiert).

Wollen Sie diese Anwendung mit der hochauflösenden Grafik auch in SCREEN 3 praktizieren, muß ich Sie an dieser Stelle leider enttäuschen, denn in SCREEN 3 ist dieser Table laut BASE-Befehl (genauer gesagt: BASE(17)) nicht belegt. Die Grafik in 64 * 48 Punkten wurde ja bereits samt aller 16 Farben mit dem Colour-Table abgedeckt.

Fassen wir also noch einmal zusammen, was sich wo in SCREEN 0 bis 3 bzgl. der hochauflösenden Grafik tut:

SCREEN 0, Anfang 2048, Länge 2048, Variable BASE(2)
SCREEN 1, Anfang 0, Länge 2048, Variable BASE(7)
SCREEN 2, Anfang 0, Länge 6144, Variable BASE(12)
SCREEN 3, Anfang /, Länge /, Variable BASE(17)

4) Nun kommen wir letztendlich zu den Sprites und ihrer Abspeicherung im Video-RAM. Glücklicherweise hat Ihr MSX-Computer eine Vielzahl von BASIC-Befehlen, die uns die Arbeit mit diesen fließend sich bewegenden Körpern erhebich vereinfacht. Die vierte BASE-Adresse bezieht sich auf die Attribute der Sprites in den einzelnen Screens.

Fangen wir mit SCREEN 0 an, müssen wir gleich zu Anfang passen: Hier gibt es keine Sprites und demnach auch keinen diesbezüglichen Attributtisch.

In SCREEN 1 sieht dies schon ganz anders aus: der Attributtisch (was das bedeutet erfahren Sie gleich anschließend in den Beispielen) beginnt bei VRAM-Adresse 6912 und ist 128 Bytes lang.

Wir gehen auf den Inhalt dieser 128 Bytes nun etwas genauer ein, können dies aber bei den weiteren Screens dann übernehmen, denn in jedem SCREEN ist die Handhabung der Sprites und des diesbezüglichen Attribut-Tisches gleich.

In jedem SCREEN (bis auf SCREEN 0) haben wir die Möglichkeit, 32 Sprites auf dem Bildschirm erscheinen zu lassen. Dies ist allerdings nur möglich, wenn wir eine Vorsichtsmaßnahme einhalten: In einer horizontalen Ebene darf die Anzahl der dargestellten Sprites niemals vier überschreiten. Passiert dies dennoch, tut sich wundersames auf dem Bildschirm: ein fünftes hinzugeommenes Sprite löscht das Sprite mit der höchsten Nummer. Ist die Vorbeigleitphase allerdings vorüber, sind alle 'alten' vier Sprites wieder wie eh und je zu erkennen.

Im Attributtisch sind zu jedem der 32 Sprites vier Informationen abgelegt (4 * 32 = 128 Bytes):

- Das erste Zeichen gibt die y-Koordinate an
- Das zweite Zeichen gibt die x-Koordinate an
- Das dritte Zeichen gibt an, welches Sprite (siehe SPRITE$-Befehl und Kapitel 5 dieses Kapitels) dargestellt werden soll
- Das vierte Zeichen gibt die Farbe des Sprites an und zudem an welcher Stelle der Ansteuerpunkt für x,y liegt

Bevor wir zur Anwendung schreiten, muß das vierte Zeichen noch ein klein wenig genauer interpretiert werden. Dazu müssen wir uns dieses Zeichen wiederum in binärer Darstellung etwas genauer betrachten: Schauen wir uns z.B. an, wie es aussieht, wenn das Sprite weiß ist (Farbe 15):

dezimal 15 = binär 00001111

Das erste, zweite, dritte und vierte Bit enthalten den Farbcode. Von rechts gesehen haben das fünfte, sechste und das siebte Bit in der binären Darstellung keine Bedeutung. Aber was ist mit Bit Nummer acht? Dazu müssen wir noch ein klein wenig tiefer in die Materie der Zahlverarbeitung bei Computern eintauchen:

In den ersten zwei Zeichen der jeweiligen Spriteattribute hatten wir bereits festgestellt, ist die y- bzw. die x-Koordinate des Sprites auf dem Bildschirm dargestellt. Wie Sie bereits wissen, hat der Bildschirm eine Auflösng in Punkten von 256*192. Kein Problem, werden Sie sagen, dann können wir unsere Sprites auch Punkt für Punkt auf dem Bildschirm gleiten lassen. Dies trifft zu, wenn Sie nur die Vertikale beobachten; das wird aber problematisch, wenn Sie die Horizonale näher betrachten: Der MSX-Computer stellt das Sprite auf dem Bildschirm dar, indem er die Ecke links oben vom Sprite als Grundkoordinate betrachtet. Es gibt keine Probleme, wenn wir das Sprite von rechts nach links gleiten lassen:

siehe Kurzlisting BASE 9 im Anhang

Es klappt wunderbar, das Sprite gleitet förmlich aus dem Bildschirm heraus und ward anschließlich nicht mehr gesehen.

Probieren wir das gleiche in der entgegengesetzten Richtung aus und das böse Erwachen setzt kurzfristig ein:

siehe Kurzlisting BASE 10 im Anhang

Zwar wunderschön, wie unser Sprite langsam von rechts in den Bildschirm hereinkommt, aber plötzlich ist Position 0,0 erreicht und ... das Sprite ist nicht nach links aus dem Bildschirm verschwunden, sondern steht dort oben links in der Ecke und wartet auf weitere Kommandos.

Nun können wir in Aktion treten, indem wir das achte Bit unseres vierten Zeichens auf 1 setzen (wir addieren 128 zu unser Farbgebung hinzu). Dies zeigt dem Computer an, in welcher Notlage wir im Augenblick stecken und so wird die Koordinatenangabe nun folgendermaßen interpretiert:

x=x-32   y=y

Ihr MSX-Computer setzt einfach seine Rechenkoordinate für die Sprite-Darstellung auf -32 und schon ist das Sprite vom Bildschirm verschwunden.

Diese komplizierte Prozedur vollführt Ihr MSX-Computer automatisch im BASIC, indem Sie ihn mit neativen Werten füttern. Nur seien Sie vorsichtig, den zu viel Futter verdirbt ihm den Appetit. Sie können die FOR ... NEXT-Schleife (Zeile 40) in den Minus-Bereich weiterführen, nur wenn dies zu weit geht, erscheint das Sprite plötzlich wieder an der rechten Bildschirmseite (ohne langsam hereinzugleiten). Diesen unerwarteten Effekt können Sie verhindern, wenn Sie nicht mehr als 32 Ziffern über den normalen Bereich hinausgehen (von -32 bis +287). Wir wollen uns aber mal interessehalber ansehen, wie kompliziert der Computer bei dieser Prozedur im VRAM arbeiten muß:

siehe Kurzlisting BASE 11 im Anhang

Soweit unser Programm mit der Darstellung von rechts nach links. Nun wollen wir aber den von-rechts-Hereingleiteffekt erzielen:

siehe Kurzlisting BASE 11a im Anhang

Neben der Attributgebung für die Sprites (Farbe, Bildschirmposition, Spritenummer im Sprite Generator Table) müssen wir auch bestimmen können, wie ein Sprite auszusehen hat. Dies ist im fünften Abschnitt des entsprechend zugeordneten Video-RAM-Bereichs abgespeichert:

5) Bis zu 32 Sprites können mit MSX-BASIC in ihrem Aussehen definiert und gleichzeitig auf den Bildschirm gebracht werden. Wie und wo werden diese Informationen im Speicher abgelegt?

Wie bereits bei den Attributen der Sprites festgestellt, verfügt SCREEN 0 über keine Sprites. Jedoch ab SCREEN 1 können Sprites definiert werden. Da die Abspeicherung der Sprites in SCREEN 1, 2 und 3 auf die gleiche Art und Weise vor sich geht, werde ich in diesem Kapitel als Orientierungsvorgabe zum Arbeiten in SCREEN 2 und 3 lediglich auf den Speicheraufbau in SCREEN 1 eingehen.

Die Abspeicherung der Sprites beginnt bei Video-RAM-Adresse 14336 und ist 2048 Speicherplätze lang. Wir können in jedem Fall mehr Sprites definieren, als wir gleichzeitig auf dem Bildschirm darstellen können: Auf dem Bildschirm können bei Beachtung der Einschränkungen (nich mehr als vier Sprites gleichzeitig in einer horizontaen Ebene, sonst überdeckt Sprite fünf Teile des Sprites mit der größten Spritenummer) 32 Sprites gleichzeitig zur Abbildung gebracht werden.

Wollen wir nur kleinere Sprites zur Abbildung bringen (8*8-Matrix), ansteuerbar durch SCREEN ,0 oder SCREEN ,1, können im Video-RAM bis zu 256 Sprites in ihrem Aussehen definiert werden. Das entspricht dem zur Verfügung stehenden Speicherbereich bei Durchführung der Rechnung 256*8 = 2048).

Da die Abspeicherungsebene für das Spriteaussehen in Speicherstelle 14336 beginnt, befindet sich Sprite 0 in den Speicherstellen 14336 bis 14343, Sprite 1 in den Speicherstellen 14344 bis 14351, bis zu Sprite 255 in den Speicherstellen 16377 bis 16384. Das Spriteaussehen kann zur leichteren Veranschaulichung am besten binär eingegeben werden aber sollte der Platzersparnis wegen anschließend in das Dezimalsystem übertragen werden:

binär		dezimal	
	00000000		0
	01111110		126
	01111110		126
	01100110		102
	01100110		102
	01111110		126
	01111110		126
	00000000		0

Mit dem BASIC-Befehl PUT SPRITE kann anschließend das erstellte Sprite auf dem Bildschirm zur Anzeige gebracht werden:

siehe Kurzlisting BASE 12 im Anhang

Nun erscheint das Sprite in der Mitte des Bildschirms; vergleichen Sie es mit unserem zeichnerischen Entwurf.

Wollen wir uns den Zwischenschritt ersparen (Umrechnung binär-dezimal), können wir auch direkt die Nullen und Einsen verarbeiten. Das Programm sähe dann folgendermaßen aus:

siehe Kurzlisting BASE 13 im Anhang

Da wir uns in diesem Kapitel nur mit dem Aufbau des Speichers an BASE orientieren wollen und dabei den luxuriösen Befehlsschatz des MSX-BASIC (gerade was die Grafik betrifft) soweit wie möglich durch direktes Speichereingreifen ersetzen wollen, soll auch das Grafikkommando PUT SPRITE in Zeile 70 bzw. in Zeile 90 durch direkten Eingriff in den Speicher simuliert werden. Wissen Sie noch, wie wir da vorzugehen haben?

Richtig, wir müssen nur die Attribute des gewünschten Sprites im Speicher ablegen. Erinnern wir uns: zu den Attributen zählen:

1) die Y-Position (in unserem Fall = 96)
2) die X-Position (in unserem Fall = 128)
3) die Spritenummer (in unserem Fall = 0)
4) die Spritefarbe sowie der mögliche Verschiebefaktor (letzteres ist in unserem Fall nicht nötig, da das Sprite in der Bildschirmmitte positioniert wird), also Farbe z.B. = weiß = 15

Da sich die Eingabe gleich auf das erste Sprite bezieht (Adresse im Attributtisch: 6912) können wir Zeile 70 bzw. Zeile 90 direkt ersetzen durch:

```
70 bzw. 90 VPOKE 6912,96
71 bzw. 91 VPOKE 6913,128
72 bzw. 92 VPOKE 6914,0
73 bzw. 93 VPOKE 6915,15
```

Wie ist aber vorzugehen, wenn wir größere Sprites darstellen wollen? In diesem Fall kennzeichnen wir diese unsere Absicht mit dem Parameter 2 oder 3 (vergrößert) im 2. Parameter des SCREEN-Befehls z.B. SCREEN ,2 oder SCREEN ,3.

Im Video-RAM-Speicher lassen sich 256 Sprites in der Größe 8*8 ablegen, jedoch brauchen wir zur Definition jedes 16*16-Sprites vier Sprites in der Größe 8*8, damit das "große" Sprite auch vollständig ist d.h. wir können nur noch 64 verschiedene Sprites abrufbereit im Speicher niederlegen (64*32=2048, das entspricht dem zur freien Verfügung stehenden Speicherraum zur Spriteabspeicherung). Trotzdem ist es möglich, bis zu 32 Sprites der Größe 16*16 auf dem Bildschirm gleichzeitig darzustellen.

Die Speicherung der Sprites im Video-RAM geht folgendermaßen vor sich: Erst wird das Teil links oben gelesen, dann das Teil links unten, schließlich die Teile rechts oben und rechts unten. Jedes Teil ist wiederum 8*8 Punkte groß. Malen wir uns doch einfach unser Muster von vorhin in der Größe 16*16 auf und rechnen gleich die entsprechenden Dezmalzahlen dazu aus:

Sprite 1:	00000000	00000000	Sprite 3	0	0
	01111111	11111110		127	254
	01111111	11111110		127	254
	01100000	00000110		96	6
	01100000	00000110		96	6
	01100000	00000110		96	6
	01100011	11000110		99	19
	01100010	01000110		98	70
Sprite 2:	01100010	01000110	Sprite 4	98	70
	01100011	11000110		99	19
	01100000	00000110		96	6
	01100000	00000110		96	6
	01100000	00000110		96	6
	01111111	11111110		127	254
	01111111	11111110		127	254
	00000000	00000000		0	0

Ganz schön groß und auch aufwendig ist es ja doch geworden. Schauen wir uns an, wie und wohin wir die Dezimalwerte in einem Programm einlesen können:

siehe Kurzlisting BASE 14 im Anhang

Schauen wir uns nun einmal an, wie wir ohne die Dezimalumrechnung und ohne Gebrauch der MSX-Grafik-BASIC-Befehle direkt unser 16*16-Sprite in den Speicher übertragen können:

siehe Kurzlisting BASE 15 im Anhang

Sicherlich werden Sie nun zu schätzen wissen, wie wichtig es doch bei der Spriteerstellung und der Spriteverwendung ist, einerseits ein luxuriöses BASIC zu besitzen (die Befehle PUT SPRITE und insbesondere die Variablenzuweisung SPRITE$ erweisen sich dabei als besonders hilfreich, andererseits aber auch den Zugriff auf einen Spriteeditor zu haben, der uns wahrlich viel Arbeit abnimmt ... und dafür ist ja gerade ein Computer besonders gut geeignet: lästige Arbeit schnell und ohne großen Aufwand zu erledigen und dem Menschen die Dinge zu überlassen, wo Überlegung, Logik und Kreativität gefragt sind (ähnlich einem Taschenrechner). Aber genauso, wie man beim selbstverständlich gewordenen Umgang mit dem Taschenrechner nicht das Kopfrechnen ergessen sollte, so sollte es auch beim Arbeiten mit dem Computer sein: trotz allem Luxus heiß es, zu verstehen, was im Inneren des Computers, in diesem Fall in seinem Video-RAM im besonderen, abläuft, denn: wir wollen die Maschine kontrollieren und überwachen und bitte nicht durch sie kontrolliert werden.

Zum Abschluß dieses Kapitels noch einmal in der Übersicht:

Die Abspeicherung der Sprites, ihre Anzahl, ihr Speicherbeginn, ihre Speicherlage und last but not least die entsprechende BASE-Variable:

SCREEN 0:
keine Sprites, BASE(4)

SCREEN 1,0 und SCREEN 1,1:
256 Srites 8*8, Anfang: 14336, Länge: 2048, BASE(9)

SCREEN 1,2 und SCREEN 1,3:
64 Sprites 16*16, Anfang: 14336, Länge: 2048, BASE(9)

SCREEN 2,0 und SCREEN 2,1:
256 Sprites 8*8, Anfang: 14336, Länge: 2048, BASE(14)

SCREEN 2,2 und SCREEN 2,3:
64 Sprites 16*16, Anfang: 14336, Länge: 2048, BASE(14)

SCREEN 3,0 und SCREEN 3,1:
256 Sprites 8*8, Anfang: 14336, Länge: 2048, BASE(19)

SCREEN 3,2 und SCREEN 3,3:
64 Sprites 16*16, Anfang 14336, Länge: 2048, BASE(19)

BASE 1

```
            -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN Ø
30 FOR N=Ø TO 19
40 PRINT N"=";BASE(N),
50 NEXT N
```

BASE 2

```
            -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 1
30 FOR N=6144+15 TO 6144+751 STEP 32
40 VPOKE N,2
50 NEXT N
```

BASE 3

```
            -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 1
30 KEY ON
40 WIDTH 2Ø
50 COLOR 15,4,5
60 FOR N=6144 TO 6911
70 VPOKE N,2
80 PRINT STRING$(2Ø,"A");
90 NEXT N
```

BASE 4

```
            -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 1
30 FOR N=8192 TO 8192+32
40 VPOKE N,INT(RND(1)*255)
50 NEXT N
```

BASE 4a

```
            -*-*-*-*-*-
21 FOR N=32 TO 255
22 PRINT CHR$(N);
23 NEXT N
```

BASE 5

```
            -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 FOR N=8192 TO 8192+6144
40 VPOKE N,INT(RND(1)*255)
50 NEXT N
```

BASE 6

```
            -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN Ø
30 FOR N=2Ø48+(65*8) TO 2Ø48+(65*8)+7
40 PRINT VPEEK(N)
50 NEXT N
```

BASE 7

```
            -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN Ø
30 FOR N=2Ø48+(65*8) TO 2Ø48+(65*8)+7
40 B$=BIN$(VPEEK(N))
50 PRINT STRING$(8-LEN(B$),"Ø")+B$
60 NEXT N
```

BASE 8

```
        -*-*-*-*-*-
1Ø COLOR 1,15
2Ø SCREEN 2
3Ø FOR N=8192 TO 8192+6143
4Ø VPOKE N,INT(RND(1)*255)
5Ø VPOKE N-8192,INT(RND(1)*255)
6Ø NEXT N
```

BASE 9

```
        -*-*-*-*-*-
1Ø COLOR 1,15
2Ø SCREEN 2,3
3Ø SPRITE$(Ø)=STRING$(32,"U")
4Ø FOR N=Ø TO 255
5Ø FOR M=Ø TO 1Ø
6Ø NEXT M
7Ø PUT SPRITE Ø,(N,Ø)
8Ø NEXT N
9Ø GOTO 9Ø
```

BASE 1Ø

```
        -*-*-*-*-*-
1Ø COLOR 1,15
2Ø SCREEN 2,3
3Ø SPRITE$(Ø)=STRING$(32,"U")
4Ø FOR N=255 TO Ø STEP -1
5Ø FOR M=Ø TO 1Ø
6Ø NEXT M
7Ø PUT SPRITE Ø,(N,Ø)
8Ø NEXT N
9Ø GOTO 9Ø
```

BASE 11

```
        -*-*-*-*-*-
1Ø COLOR 1,15
2Ø SCREEN 2,3
3Ø SPRITE$(Ø)=STRING$(32,"U")
4Ø FOR N=Ø TO 255
5Ø VPOKE 6912,Ø
6Ø VPOKE 6913,N
7Ø VPOKE 6914,1
8Ø VPOKE 6915,1
9Ø NEXT N
1ØØ GOTO 1ØØ
```

BASE 11a

```
        -*-*-*-*-*-
1Ø COLOR 1,15
2Ø SCREEN 2,3
3Ø SPRITE$(Ø)=STRING$(32,"U")
4Ø FOR N=255 TO Ø STEP -1
5Ø VPOKE 6912,Ø
6Ø VPOKE 6913,N
7Ø VPOKE 6914,1
8Ø VPOKE 6915,1
9Ø NEXT N
1ØØ GOTO 1ØØ
```

BASE 12

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 1,0
30 FOR N=0 TO 7
40 READ A
50 VPOKE 14336+N,A
60 NEXT N
70 PUT SPRITE 0,(128,96)
80 GOTO 80
90 DATA 0,126,126,102,102,126,126,0
```

BASE 13

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 1,0
30 FOR N=0 TO 7
40 READ A$
50 A$="&B"+A$
60 A=VAL(A$)
70 VPOKE 14336+N,A
80 NEXT N
90 PUT SPRITE 0,(128,96)
100 GOTO 100
110 DATA 00000000
120 DATA 01111110
130 DATA 01111110
140 DATA 01100110
150 DATA 01100110
160 DATA 01111110
170 DATA 01111110
180 DATA 00000000
```

BASE 14

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 1,2
30 FOR N=0 TO 3
40 FOR M=0 TO 7
50 READ A
60 VPOKE 14336+(N*8)+M,A
70 NEXT M
80 NEXT N
90 VPOKE 6912,96
100 VPOKE 6913,128
110 VPOKE 6914,1
120 VPOKE 6915,1
130 GOTO 130
140 DATA 0,127,127,96,96,96,99,98
150 DATA 98,99,96,96,96,127,127,0
160 DATA 0,254,254,6,6,6,198,70
170 DATA 70,198,6,6,6,254,254,0
```

BASE 15

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 1,2
30 FOR N=0 TO 15
40 READ A$
50 B$="&B"+LEFT$(A$,8)
60 VPOKE 14336+N,VAL(B$)
70 NEXT N
80 RESTORE
90 FOR N=0 TO 15
100 READ A$
110 B$="&B"+RIGHT$(A$,8)
120 VPOKE 14352+N,VAL(B$)
130 NEXT N
140 VPOKE 6912,96
150 VPOKE 6913,128
160 VPOKE 6914,1
170 VPOKE 6915,1
180 GOTO 180
190 DATA 0000000000000000
200 DATA 0111111111111110
210 DATA 0111111111111110
220 DATA 0110000000000110
230 DATA 0110000000000110
240 DATA 0110000000000110
250 DATA 0110001111000110
260 DATA 0110001001000110
270 DATA 0110001001000110
280 DATA 0110001111000110
290 DATA 0110000000000110
300 DATA 0110000000000110
310 DATA 0110000000000110
320 DATA 0111111111111110
330 DATA 0111111111111110
340 DATA 0000000000000000
```

Befehle Grafik VPEEK

Abfrage des Inhalts von Speicherstellen im Video-RAM (VRAM)

siehe hierzu auch die Programme im Anhang: VPEEK 1
VPEEK 2
VPEEK 3
VPEEK 4
VPEEK 5
VPEEK 6
VPEEK 7
Programm 12

Im Gegensatz zu vielen anderen Computern unterscheiden wir bei den MSX-Geräten nicht nur zwischen RAM und ROM, sondern auch zwischen RAM und VRAM. Soviel zur Erklärung dieser Begriffe: Im ROM ist das BASIC und das Betriebssystem der MSX-Computer abgelegt. Dieser sogenannte Festspeicher umfaßt bei den MSX-Computern 32 Kilobyte und besteht großenteils aus aneinandergereihten Maschinensprachroutinen, die den MSX-BASIC-Kommandos zur Ausführung verhelfen. Das RAM ist der Schreib-Lese-Speicher. Dieses RAM ist bei den MSX-Computern unterschiedlich groß. Im RAM werden nicht nur die Programme abgelegt, hier merkt sich unser MSX-Computer auch Dinge, die für uns gar nicht so offensichtlich sind wie ein Programmlisting: Die Variablen, die Belegung der Funktionstasten, die aktuelle Vorder-, Hintergrund- und Umrandungsfarbe usw.

Getrennt von diesen zwei Speicherblöcken haben die MSX-Computer noch einen davon getrennten Schreib-Lesespeicher, der aber nur für Grafik zuständig ist. Hier wird das Aussehen von Buchstaben und Zeichen, die gesamte grafische aber auch die textliche Bildschirmdarstellung, das Spriteaussehen sowie die aktuelle Spritepositionierung abgelegt. Nicht nur die grafische Punktdarstellung, nein, auch die farbliche Zuordnung der Farbmuster finden hier ihren Platz und Raum. Der VRAM-Speicher ist bei den MSX-Computern 16 KB groß.

Während wir mit dem Befehl PEEK im RAM zw. ROM lesen können, verwenden wir den Befehl VPEEK entsprechend zum Lesen des VRAM. Da das VRAM 16 KB groß ist, können wir mit dem VPEEK-Befehl auch 'nur' die Speicherstellen zwischen 0 und 16383 nach ihrem Inhalt befragen. Bei größeren oder kleineren Zahlen erscheint auf dem Bildschirm die Fehlermeldung: 'Illegal function call'.

In diesem Kapitel wollen wir uns nicht tiefgehend mit den genauen Positionierungen im VRAM auseinandersetzen. Dafür sind wir bereits bei der Behandlung des BASE-Befehls näher darauf eingegangen. Vielmehr wollen wir uns in diesem Kapitel nur einmal an einem Beispiel verdeutlichen, wie und wo im VRAM das Aussehen unserer Buchstaben und Zeichen auf SCREEN 0 abgelegt ist und wie wir eine diesbezügliche Abfrage der Speicherstellen des VRAM mit VPEEK durchführen können:

siehe Kurzlisting VPEEK 1 im Anhang

Ein klein wenig abstrakt mutet es ja schon an, wenn Sie nun für den Buchstaben 'A' als Aussehensbeschreibung die folgenden 8 Ziffern ausgegeben bekommen:

32, 80, 136, 136, 248, 136, 136, 0

Wir müssen noch eine Stufe tiefer einsteigen, um wahrlich hinter diesen Zahlen die Form unseres 'A's wieder zu erkennen. Zu diesem Zweck müssen wir uns der Tatsache bewußt werden, daß unser MSX-Computer genau besehen eigentlich nichts anderes versteht als nur die Zustände '0' und '1'. Da die Rechnerarbeit mit diesen zwei Zuständen aber so schnell im Computerhirn abläuft (bis zu weit über 100000mal pro Sekunde bei Ihrem MSX-Computer) und bestimmte an-aus- bzw. 1-0-Zustände auch folgerichtig als sinnvoll von unserem Computer erkannt werden, wird er halt durch Schnelligkeit schlau gemacht und die angebliche Dummheit flimmert gar nicht mehr hindurch.

Mit dem Befehl BIN$ können wir uns die Zahlen von vorher in diese 0-1-Darstellung umwandeln:

siehe Kurzlisting VPEEK 2 im Anhang

Mit ein klein wenig Phantasie erkennen Sie nun bereits Punkt für Punkt bzw. 1 für 1 unser 'A' wieder. Machen wir uns dies noch ein klein wenig deutlicher, indem wir den unvollständigen Ausgabezeilen entsprechend viele Nullen voranstellen (denn jede Dezimalzahhl zwischen 0 und 255 läßt sich mit einer achtstelligen 0-1-Zahl oder Binärzahl darstellen):

siehe Kurzlisting VPEEK 3 im Anhang

Wenn Sie sich nun die Nullen wegdenken, sehen Sie Punkt für Punkt den genauen Aufbau unseres 'A's. Wie bitte? Noch deutlicher?

siehe Kurzlisting VPEEK 4 im Anhang

Schauen Sie einmal im Anhang nach, wie wir mit Programm 12 den gesamten Zeichensatz Ihres Computers ausgedruckt haben. Der eigentliche Vorgang ist dort genau der gleiche, wie wir ihn in Programm VPEEK 4 gerade kennengelernt haben:

siehe Programm 12 im Anhang

Auf eine ähnliche Art und Weise können wir uns das Sprite Punkt für Punkt oder aber die gesamte grafische Darstellung auf dem Bildschirm im VRAM-Speicher anschauen. Doch stellen Sie sich gerade die Abfrage der Speicherstellen des VRAM bei SCREEN 2 bitte nicht zu einfach vor, denn Ihr MSX ist ein Farbcomputer und, soviel sei hier noch angemerkt, die Grafik und die Farbe, die Ihnen auf dem Bildschirm als Eins erscheinen, belegen im VRAM zwei verschiedene Speicherbereiche. Daraus folgt dann erst die eigentliche Farbgrafik. Hingegen sollten Sie mit unserem Programmen VPEEK 1 bis VPEEK 4 ruhig einmal im VRAM weiterforschen, wo Sprites und Characterzeichen in ihrem Aussehen abgespeichert sind (die Aufteilung des VRAM können Sie bei der Beschreibung des BASE-Kommandos erfahren).

Zuletzt wollen wir noch einen Trick verraten: Wie kann man durch das Lesen im VRAM eine Eingabe zehnfach darstellen? Geben Sie dazu folgendes Programm ein:

siehe Kurzlisting VPEEK 5 im Anhang

Nun lesen wir vom Bildschirm und schreiben an verschiedene Bildschirmpositionen das, was wir gelesen haben.

siehe Kurzlisting VPEEK 6 im Anhang

Wir können mit VPEEK gar die Funktionstastenlatte vom Bildschirm ablesen und an einer anderen Stelle wieder erscheinen lassen:

siehe Kurzlisting VPEEK 7 im Anhang

VPEEK 1

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 0
30 FOR N=1 TO 8
40 PRINT VPEEK(2048+65*8+N-1);
50 NEXT N
```

VPEEK 2

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 0
30 FOR N=1 TO 8
40 PRINT BIN$(VPEEK(2048+65*8+N-1))
50 NEXT N
```

VPEEK 3

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 0
30 FOR N=1 TO 8
40 PRINT STRING$(8-LEN(BIN$(VPEEK(2048+6
5*8+N-1))),"0")+BIN$(VPEEK(2048+65*8+N-1
))
50 NEXT N
```

VPEEK 4

```
        -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN Ø
30 FOR N=1 TO 8
40 A$=STRING$(8-LEN(BIN$(VPEEK(2Ø48+65*8
+N-1))),"Ø")+BIN$(VPEEK(2Ø48+65*8+N-1))
5Ø FOR M=1 TO 8
6Ø IF MID$(A$,M,1)="1" THEN PRINT CHR$(2
19);ELSE PRINT " ";
7Ø NEXT M
8Ø PRINT
9Ø NEXT N
```

VPEEK 5

```
        -*-*-*-*-*-
1Ø COLOR 1,15
2Ø SCREEN Ø
3Ø WIDTH 4Ø
4Ø A$="MSX-Computer"
5Ø PRINT A$
6Ø FOR M=1 TO 1Ø
7Ø FOR N=1 TO LEN(A$)
8Ø VPOKE M*2Ø+N,VPEEK(N-1)
9Ø NEXT N
1ØØ NEXT M
11Ø LOCATE Ø,1Ø
```

VPEEK 6

```
        -*-*-*-*-*-
1Ø COLOR 1,15
2Ø SCREEN Ø
3Ø WIDTH 4Ø
4Ø A$="MSX-Computer"
5Ø PRINT A$
6Ø FOR M=1 TO 1Ø
7Ø FOR N=1 TO LEN(A$)
8Ø VPOKE INT(RND(1)*94Ø+2Ø),VPEEK(N-1)
9Ø NEXT N
1ØØ NEXT M
11Ø LOCATE Ø,1Ø
```

VPEEK 7

```
        -*-*-*-*-*-
1Ø COLOR 1,15
2Ø SCREEN Ø
3Ø WIDTH 4Ø
4Ø FOR M=1 TO 1Ø
5Ø FOR N=1 TO 4Ø
6Ø VPOKE M*4Ø+N,VPEEK(23*4Ø+N-1)
7Ø NEXT N
8Ø NEXT M
```

Befehle Grafik VPOKE

Schreiben von Zahlwerten in die Speicherstellen des VRAM

siehe hierzu auch die Programme im Anhang: VPOKE 1
VPOKE 2
VPOKE 3
VPOKE 4
VPOKE 5
VPOKE 6
VPOKE 7
Programm 2
Programm 3
Masch.sprach.grafik

Wie jeder andere Computer besitzt auch Ihr MSX-Computer zwei unterschiedliche Speicher: Einmal den ROM, das ist der nur lesbare Speicher (Read Only Memory), in dem u.a. das BASIC enthalten ist aber auch die Grunddefinitionen des Zeichenaussehens und die ursprüngliche Belegung der Funktionstasten. Außerdem gibt es den RAM (Random Access Memory), der all die von Ihnen eingegbenen Mitteilungen, seien es nun Inhalte von Variablen oder aber Programmzeilen, so lange festhält, bis Sie den Computer wieder ausschalten. Der RAM ist also frei beschreibbar und auch lesbar, jedoch bleiben die Informationen nur so lange erhalten, wie der Computer mit Strom versorgt ist (aus diesem Grunde gibt es Magnetspeicher wie Kassette und Diskette, die diese 'flüchtigen' Informationen auf Dauer festhalten).

Wollen wir im RAM oder ROM lesen, bedienen wir uns des MSX-BASIC-Kommandos PEEK. Wir geben dazu noch die entsprechende Speicherstelle (bei einem 8-Bit-Homecomputer wie dem Ihrigen MSX-Computer erstrecken sich die abfragbaren Speicherplätze im Höchstfall von 0 bis 65535) und veranlassen den Bildschirmausdruck durch ein vorangestelltes PRINT.

Wollen Sie hingegen in den Speicher etwas hineinschreiben, benötigen Sie dazu den Befehl POKE (man spricht deshalb auch immer so gerne vom 'wilden HerumPOKEn im Speicher'); jedoch, wie bereits bei der Unterscheidung von ROM und RAM erwähnt: Sie können nur in den beschreibbaren Speicher = das RAM hineinschreiben, beim POKEn in den ROM-Bereich bleibt dies ohne Wirkung.

Im Gegensatz zu den meisten anderen Homecomputern weisen die MSX-Computer bzgl. des RAM eine Besonderheit auf: Einerseits haben sie freien RAM-Speicherplatz, der zur Aufnahme von Variablen und Programmen dient. Andererseits ist da aber noch ein vom RAM vollkommen getrennter Speicherbereich, der nur für Dinge verantwortlich zeichnet, die mit der Bildschirmdarstellung zu tun haben. Dieser Speicher ist bei den MSX-Computern 16 Kilobyte groß (mehr als 16000 Speicherplätze) und enthält Informationen (je nach Anwahl des SCREEN) über die Zeichendarstellung auf dem Bildschirm, das punktweise Aussehen der dargestellten Zeichen, Informationen über den Aufbau von bis zu 256 verschiedenen Sprites, den aktuellen Standort der Sprites, die punktweise Farbgrafik usw.

Da dieser RAM-Speicher mit Bildschirm=Videoaufbau zu tun hat, heißt er nicht nur einfach RAM sondern VRAM (Video-RAM). Sie können sämtliche Tätigkeiten, die Sie mit jedbeliebigen Grafikbefehlen ausführen, auch direkt in den VRAM hineinPOKEn, jedoch gehört eine detaillierte Kenntnis des VRAM-Aufbaus dazu, die Sie sich im Kapitel zum BASE-Befehl suksessive aneignen können.

Ich sprach gerade vom POKEn im VRAM. Dies geht aufgrund des vom normalen RAM getrennten Video-RAM-Speichers nicht mit dem Kommando PEEK und POKE vor sich, sondern mit den Extra-MSX-Kommandos VPEEK (bzw. zum Bildschirmausdruck: PRINT VPEEK(N)) und VPOKE (bzw. zur Eingabe: VPOKE N,M) vor sich. VPEEK liest dabei Informationen aus dem VRAM, während VPOKE das Schreiben bzw. Speichern von Informationen im VRAM ermöglicht.

Schauen wir uns die Arbeit mit VPEEK und VPOKE an zwei kleinen Beispielen etwas genauer an:

siehe Kurzlisting VPOKE 1 im Anhang

Dieses Programm soll wohlgemerkt nicht den Aufbau des VRAM erklären, sondern vielmehr die Bedeutung des VPEEK-Befehls verdeutlichen: Zuerst wurde das MSX-Logo links oben auf den Bildschirm geschrieben. Wir setzen hier als bekannt voraus, daß der Bildschirmaufbau in SCREEN 0 bei Speicherstelle 0 im VRAM beginnt (siehe unter BASE). Wir schauten uns im Ablauf dieses Programms die ersten drei Zeichen im VRAM etwas genauer an:

```
PRINT VPEEK(0): 77 = M
PRINT VPEEK(1): 83 = S
PRINT VPEEK(2): 88 = X
```

Die drei Speicherstellen tragen wahrlich die Information über die drei Buchstaben 'MSX', die wir vorher mit PRINT "MSX" in die erste Bildschirmzeile geschrieben hatten. Um dies zu verdeutlichen POKEn wir sie in einem zweiten Programm einfach mit VPOKE nach dem Löschen des Bildschrms wieder an die drei ersten Positionen des VRAM:

siehe Kurzlisting VPOKE 2 im Anhang

Sie werden sich sicherlich wundern, wie für die Buchstaben 'M' und 'S' und 'X' die Zahlwerte '77' und '83' und '88' zustandekommen. Dies hängt damit zusammen, daß jegliche auf dem Bildschirm dargestellten Zeichen als Zahlwerte verschlüsselt werden z.B. 32 für das Leerzeichen (siehe dazu auch unter den BASIC-Befehlen ASC und CHR$ in diesem Buch nach). So müssen wir, um Buchstaben auf dem Bildschirm mit VPOKE zu plazieren, die ASC-Codes jedes Zeichens eingeben. Schauen wir uns dies einmal im folgenden Programm an, wo auf diese Art und Weise ein von Ihnen eingegebener Satz Zeichen für Zeichen suksessive auf dem Bildschirm SCREEN 0 in der linken oberen Ecke plaziert wird:

siehe Kurzlisting VPOKE 3 im Anhang

Sie werden sich sicherlich nun fragen: Warum so kompliziert, wenn es doch viel einfacher mit PRINT "..." geht. Die ausführliche Vorstellung des VPOKEns soll Ihnen aus drei Gründen nahegebracht werden:

1. Der PRINT-Befehl arbeitet in seinen Einzel-Maschinenbefehlen geauso kompliziert, wie gerade teilweise dargestellt. Dadurch, daß Sie den VPOKE-Befehl hier genauer kennenlernen, schauen Sie auch mal ein klein wenig tiefer in die Komplexität der kleinen Schritte bei der Maschinensprache hinein, die aber in Wirklichkeit noch viel feinschrittiger arbeitet - die direkte Ansprache des VRAM ist eigentlich bereits ihr Zielpunkt.

2. Wir sprechen hier über den VPOKE-Befehl und Sie sollen diesen Befehl nicht nur theoretisch zur Notitz nehmen, sondern auch praktisch begreifen lernen.

3. Es gibt kein BASIC-Kommando, das Ihnen das Lesen vom Textbildschirm erleichtert (wie im Gegensatz dazu auf den Grafikbildschirmen der Befehl POINT). Wollen Sie dies, müssen Sie sich zumindest mit dem unkomfortabel anmutenden VPEEK-Befehl wohl oder übel anfreunden.

Bevor Sie nun sicherlich wie wild im VRAM herumzuPOKEn beginnen, hier noch einmal der Vorsichtsaufruf: Lernen Sie erst den Aufbau des VRAM durch den BASE-Befehl näher kennen, ansonsten kann bereits durch einige unwillkürlich gesetzte VPOKEs Verheerendes passieren:

siehe Kurzlisting VPOKE 4 im Anhang

Viele Buchstaben haben zu wuchern begonnen, fast ist nichts mehr lesbar und man fühlt sich veranlaßt, den Computer aus- und wieder einzuschalten; doch es geht einfacher: löschen Sie den Bildschirm bzw. den VRAM-Speicher mit einem SCREEN-Befehl. Dadurch wird das VRAM wieder in den Anfangszustand versetzt und alles ist wieder in bester Ordnung.

Dennoch wollen wir uns (allerdings mit Vorsicht) eine zweite direkt sichtbare Anwendung des VPOKE-Befehls auf SCREEN 0 anschauen: Wir wollen ein Zeichen, in diesem Fall das Leerzeichen, in seinem Aussehen überall auf dem Bildschirm so verändern, daß es wie ein kleines Schachbrett aussieht d.h. der Bildschirm besteht somit genau betrachtet aus fast 1000 Kästchen zu je 64 Feldern (genaugenommen sind es nur jeweils 48 Felder, weil auf SCREEN 0 'nur' 3/4 der Characterzeichen dargestellt werden - siehe unter SCREEN):

siehe Kurzlisting VPOKE 5 im Anhang

Wie ist das geschehen? Die Speicherstellen des Leerzeichenaussehens auf SCREEN 0 beginnen bei der VRAM-Speicherstelle 2048+32*8 und setzen sich bis 2048+32*8+7 fort. Warum 32*8? Wie ich bereits vorhin andeutete, ist das Leerzeichen gleichbedeutend mit dem ASC-Code 32. Die VRAM-Adresse 2048 ist der Anfang des VRAM-Bereichs, der für das Aussehen sämtlicher darstellbarer Zeichen verantwortlich ist. Also müssen wir die ersten 32*8 Speicherstellen überspringen (32*8 deshalb, weil die Zählweise bei Zeichen 0 beginnt), bis wir bei unserem Zeichen 32 angelangt sind. Danach werden dieser und die folgenden 7 Werte mit 85 und 170 belegt. Warum ausgerechnet diese Zahlen, erläutert uns die Aufschlüsselung durch den BIN$-Befehl - wohlgemerkt die kleinste Zahlsystemdarstellung für unseren MSX-Computer (nicht wie im Dezimalsystem sind hier 10 Zustände gültig, sondern lediglich 2 (ein und aus bzw. 0 und 1)):

```
BIN$(85)= 01010101
BIN$(170)=10101010
```

Überall dort, wo eine 1 geschrieben steht, ist im Leerzeichen von nun an ein Bildschirmpunkt gesetzt. Umgekehrt bleibt bei einer binären 0 der Bildschirmpunkt an der entsprechenden Stelle unbesetzt.

Keine Angst, Sie können selbstverständlich leicht den Urzustand des Leerzeichens wiederherstellen:

1. indem Sie irgendein SCREEN N-Kommando eingeben

2. indem Sie unser obiges Programm mit folgender Zeilenänderung noch einmal ablaufen lassen (wir POKEn ganz einfach an die soeben geänderten Bildschirmpositionen einfach wieder eine 0):

```
40 VPOKE 2048+32*8+N,0
50 VPOKE 2048+32*8+N+1,0
```

Vorhin wies ich bereits darauf hin, daß das VPOKEn sehr wichtig für diejenigen ist, die in Maschinensprache arbeiten können und somit nicht mit BASIC-Kommandos wie PRINT umgehen dürfen. Um Ihnen aufzuzeigen daß VPOKE zur Bildschirmdarstellung in BASIC nicht sinnvoll ist, sei folgender Geschwindigkeitstest durchgeführt:

siehe Kurzlisting VPOKE 6 im Anhang

Die Ergebnisse dieses kleinen Programms müßten auf allen MSX-Computern gleich sein:

1. Beschreiben von SCREEN 0 mit PRINT:
   3.36 Sekunden

2. Beschreiben von SCREEN 0 mit VPOKE:
   3.64 Sekunden

3. Beschreiben von SCREEN 0 mit PRINT STRING$:
   0.58 Sekunden

4. Beschreiben von SCREEN 0 mit Maschinenroutine:
   0.02 Sekunden

Ui, was für ein Unterschied. Im Kapitel zur grafischen Maschinensprachprogrammierung werde ich darauf näher eingehen. Für hier soll diese Demonstration lediglich als Beweis für die Schnelligkeit der Maschinensprache und die langsame Abarbeitung des VPOKE-Kommandos gelten.

Nun abschließend noch ein paar kurze Hinweise:

1. Soweit Sie sich im VRAM auskennen, können Sie 16 Kilobyte Speicherstellen mit VPEEK und VPOKE bearbeiten, das sind genaugenommen die Speicherstellen 0 bis 16383. Die Eingabe hinter dem VPOKE-Befehl darf nicht kleiner als 0 und nicht größer als 255 sein. Ist die durch VPOKE angesprochene Speicherstelle bzw. der dort einzugebende Wert größer oder kleiner als erlaubt, erfolgt die Errormeldung 'Illegal function call'.

2. Da Sie unter SCREEN 0 nicht die gesamten 16 KB VRAM für die Bildschirmdarstellung benutzen, können Sie bei gewisser Vorsicht den restlichen frei gebliebenen Speicher als Variablen- oder als weiteren Bildschirmspeicher verwenden. Schauen Sie diesbezüglich bitte im Anhang oder unter dem Kommando BASE nach, wo in dem von Ihnen angewählten SCREEN noch VRAM-Speicherplatz frei ist. Vorsicht gilt jedoch besonders deshalb, weil Sie mit einem SCREEN-Befehl den gesamten VRAM-Speicher und damit auch Ihre Daten unwiderbringlich löschen.

3. Ein eleganter Effekt läßt sich mit VPOKE erzielen, indem Sie sonst durch BASIC nicht direkt ansprechbaren VRAM-Speicher als Rahmen gestalten: So lassen sich z.B. die Funktionstasten und der eventuell rechts und links vorhandene Rand (bei WIDTH kleiner als Extremwert) zum Rahmen umfunktionieren, wie folgendes Beispiel zeigt:

siehe Kurzlisting VPOKE 7 im Anhang

Hüten Sie sich aber davor, anschießend CLS, einen SCREEN-Befehl, eine andere WIDTH oder aber KEY OFF bzw. KEY ON einzugeben: der Bildschirmrahmen wäre augenblicklich zumindest teilweise wieder zerstört.

4. Auch wenn Ihnen der Aufbau des VRAM bekannt ist, werden Sie noch große Schwierigkeiten verspüren, einfach so durch VPOKE beispielsweise eine Linie in hochauflösender Grafik zu erstellen. Dies hängt einerseits damit zusammen, daß Ihr MSX-Computer sparsam mit dem Speicherplatz umgeht (ähnlich der 0-1-Darstellung von 8 Punkten durch eine in Binärdarstellung umgerechnete Dezimalzahl im Bereich von 0 bis 255 bei dem oben beschriebenen Definieren des Zeichenaussehen). Außedem sind grafische Darstellung und Farbe in zwei voneinander getrennten VRAM-Teilen aufgehoben. Gar nicht so einfach, diese zwei Bereiche sinnvoll miteinander in Verbindung zu bringen.

VPOKE 1

```
              -*-*-*-*-*-
10 SCREEN Ø
20 WIDTH 4Ø
3Ø COLOR 1,15
4Ø PRINT "MSX"
5Ø PRINT
6Ø FOR N=Ø TO 2
7Ø PRINT VPEEK(N);
8Ø NEXT N
```

VPOKE 2

```
              -*-*-*-*-*-
1Ø SCREEN Ø
2Ø WIDTH 4Ø
3Ø COLOR 1,15
4Ø VPOKE Ø,77
5Ø VPOKE 1,83
6Ø VPOKE 2,88
7Ø LOCATE Ø,1
```

VPOKE 3

```
              -*-*-*-*-*-
1Ø SCREEN Ø
2Ø WIDTH 4Ø
3Ø COLOR 1,15
4Ø PRINT
5Ø PRINT
6Ø INPUT "Bitte einen kurzen Satz eingeb
en:          ";A$
7Ø PRINT
8Ø PRINT "Nun wird der Satz in den VRAM
          ab Speicherstelle Ø gePOKEd"
9Ø FOR N=1 TO 1ØØØ
1ØØ NEXT N
11Ø FOR N=1 TO LEN(A$)
12Ø VPOKE N-1,ASC(MID$(A$,N,1))
13Ø NEXT N
```

VPOKE 4

```
-*-*-*-*-*-
10 SCREEN 0
20 WIDTH 40
30 COLOR 1,15
40 FOR N=1 TO 1000
50 VPOKE 2048+INT(RND(1)*2048),INT(RND(1
)*255)
60 NEXT N
70 LIST
```

VPOKE 5

```
-*-*-*-*-*-
10 SCREEN 0
20 COLOR 1,15
30 FOR N=0 TO 7 STEP 2
40 VPOKE 2048+32*8+N,170
50 VPOKE 2048+32*8+N+1,85
60 NEXT N
```

VPOKE 6

```
-*-*-*-*-*-
10 CLEAR 200,50000!
20 KEY OFF
30 SCREEN 0
40 WIDTH 40
50 COLOR 1,15
60 TIME=0
70 FOR N=1 TO 960
80 PRINT "*";
90 NEXT N
100 A=TIME
110 CLS
120 TIME=0
130 FOR N=0 TO 959
140 VPOKE N,42
150 NEXT N
160 B=TIME
170 CLS
180 TIME=0
190 FOR N=1 TO 24
200 PRINT STRING$(40,"*");
210 NEXT N
220 C=TIME
230 CLS
240 FOR N=1 TO 12
250 READ A$
260 POKE 49999!+N,VAL("&H"+A$)
270 NEXT N
280 DEFUSR=50000!
290 TIME=0
300 D=USR(0)
```

```
310 D=TIME
320 DATA 3e,2a,21,00,00,01,c0,03,cd,56,0
0,c9
330 CLS
340 PRINT "So lange hat es in Sekunden g
edauert:"
350 PRINT
360 PRINT "mit PRINT:";:PRINT USING "##.
##";A/50
370 PRINT "mit VPOKE:";:PRINT USING "##.
##";B/50
380 PRINT "mit PRINT STRING$:";:PRINT US
ING "##.##";C/50
390 PRINT "mit Maschinencode:";:PRINT US
ING "##.##";D/50
```

-*-*-*-*-*-

VPOKE 7

```
10 SCREEN 0
20 KEY ON
30 COLOR 1,15
40 WIDTH 34
50 FOR N=0 TO 967
60 VPOKE N,ASC("*")
70 NEXT N
80 LOCATE 0,23
90 FOR N=1 TO 24
100 PRINT
110 NEXT N
```

Befehle Grafik CLS

Löschen der Bildschirmdarstellung auf verschiedenen SCREENs

siehe hierzu auch die Programme im Anhang: CLS 1
COLOR 4

Mit diesem BASIC-Kommando kann in sämtlichen SCREENs die Bildschirmdarstellung gelöscht werden. Dabei bleibt die Funktionstastenanzeige (wenn mit KEY ON zuvor angeschaltet) sowie der Cursor (wenn im Normalzustand belassen bzw. mit LOCATE ,,1 zuvor wieder zur Anzeige gebracht) erhalten, während sonstige Bildschirmanzeigen gelöscht werden.

Man kann durch Verwendung des SCREEN-Befehls zwar auch den Bildschirm löschen, jedoch passiert dabei mehr: dies kann bei Zeiten unbeabsichtigt zu Schwierigkeiten führen, denn der gesamte Video-RAM wird in seinen Urzustand zurückversetzt, d.h. eventuell vorher umdefinierte Characterzeichen werden wieder normal dargestellt, mit Farbe unterschiedlich belegte Acht-Zeichengruppen in SCREEN 1 (siehe unter BASE) werden in der ursprüglich gewählten Vorder- bzw. Hintergrundfarbe dargestellt usw.

Somit geschieht zwar auf den ersten Blick dasselbe, wenn wir den SCREEN-Befehl oder das CLS-Kommando zum Löschen des Bildschirms verwenden, aber Vorsicht ist geboten.

Eine andere Möglichkeit der Bildschirmlöschung wird durch die Eingabe von Leerzeilen mit Hilfe des PRINT-Kommandos (gilt nur für die Textbildschirme) erzeugt. Man sollte jedoch in diesem Fall lieber auf das CLS (CLEAR SCREEN)-Kommando zurückgreifen, denn es arbeitet sehr viel schneller.

Während die COLOR-Anweisung in SCREEN 0 und 1 eine augenblickliche Reaktion herbeiführt, soweit es die Änderung der Hintergrundfarbe betrifft (2.Parameter), ist die alleinige COLOR-Eingabe zur Farbänderung auf den Grafikbildschirmen nicht ausreichend. Wir können die Hintergrundfarbe erst durch den CLS-Befehl ändern:

siehe Kurzlisting CLS 1 im Anhang

Übrigens:

1) Wenn wir uns dies auch auf die anderen SCREENs bezogen anwenden wollen, so gilt: CLS übermalt den Bildschirm in der zuletzt angegebenen Hintergrundfarbe (2.Parameter im COLOR-Befehl).

2) Wir können nicht den gleichen Effekt im Direktmodus erzeugen, indem wir anstelle des Kommandos CLS die Taste CLS/HM einmal drücken oder im Programmmodus PRINT CHR$(12) eingeben.

siehe Kurzlisting COLOR 4 im Anhang

Bei jedem SCREEN-Wechsel wird ein CLS automatisch ausgeführt. Wir können dies umgehen (SCREENwechsel ohne Bildschirmlöschung), indem wir Routinen im ROM direkt aufrufen:

```
SCREEN 0: &H0078
SCREEN 1: &H007B
SCREEN 2: &H007E
SCREEN 3: &H0081
```

Wenn dies auch jetzt noch ein wenig unklar erscheint, geben Sie zur Ausführung folgendes direkt ein:

```
SCREEN 0
DEFUSR=&H007E
A=USR(0)
```

Sie können nun Text direkt von der Tastatur eingeben, der als Grafik verschlüsselt auf dem Bildschirm erscheint. Sie können selbstverständlich auch den Bldschirm löschen durch CLS oder aber durch Eingabe von SCREEN 0 wieder in den normalen Textbildschirm zurückkehren. Bei Ausführung des SCREEN-Befehls wird der Bildschirm aber wieder gelöscht; machen Sie es doch ohne Bildschirmlöschen d.h. durch Aufruf der Maschinenroutine für SCREEN 0 und geben Sie im Direktmodus ein:

```
DEFUSR=&H0078
A=USR(0)
```

Nun sind wir zwar wieder im Ausgangsstadium, aber Sie können genau erkennen (nun auch lesbar), was Sie gerade alles 'blind' eingegeben haben.

CLS 1

-*-*-*-*-*-

```
10 SCREEN 2
20 COLOR 15,1
30 DRAW "BM 0,0"
40 OPEN "GRP:" FOR OUTPUT AS #1
50 PRINT #1,"   Nur Aenderung der
        Vordergrundfarbe"
60 FOR N=1 TO 1000
70 NEXT N
80 CLS
90 PRINT #1,"   Jetzt durch CLS auch Aen
de-      rung der Hintergrundfarbe"
100 FOR N=1 TO 1000
110 NEXT N
120 COLOR 1,4
```

Befehle Grafik KEY ON/OFF

Anstellen und Ausstellen der Funktionstastenanzeige

siehe hierzu auch die Programme im Anhang: KEY ON 1
KEY ON 2
KEY ON 3

Wenn wir unseren MSX-Computer anstellen, befinden wir uns in SCREEN 0, einem der reinen Textbildschirme. Je Zeile sind 37 Zeichen abgebildet (wir können mit WIDTH auf 40 Zeichen/Zeile erweitern) und 23 der 24 Bildschirmzeilen können direkt durch Tastatureingabe beschrieben werden. Wollen wir etwas in Zeile 24 schreiben, scrollt der Bildschirm automatisch um eine Zeile nach oben, denn die Funktionstastenanzeige ist in den Bildschirmrahmen integriert und läßt ein Überschreiben nicht zu.

Brauchen wir mehr Platz auf dem Bildschirm, können wir die Funktionstastenanzeige (Bildschirmzeile 24) selbstverständlich auch verschwinden lassen. Für diesen Zweck steht uns der MSX-Befehl KEY OFF zur Verfügung. Wollen wir die Funktionstastenanzeige wieder sehen, geben wir nun wieder KEY ON ein und alles ist beim alten:

KEY ON = 23 Textzeilen
KEY OFF = 24 Textzeilen

Während uns im Normalfall in der Funktionstastenanzeige die Belegung der Funktionstasten 1 bis 5 angezeigt wird, ändert sich die Darstellung beim gleichzeitigen Niederdrücken einer der zwei SHIFT-Tasten (nun wird die Belegung der Funktionstasten 6 bis 10 angezeigt).

Diese Anzeige ist allerdings bei weitem nicht vollständig, denn jede Funktionstaste kann ja mit bis zu 16 Zeichen belegt werden. Stellen wir SCREEN 0 auf die größtmögliche Bildschirmweite ein (WIDTH 40), wird uns in der Funktionstastenlatte nur Zeichen 1 bis 7 jeder Funktionstaste angezeigt:

```
1234567  1234567  1234567  1234567  1234567
  F1       F2       F3       F4       F5
```

Gehen wir gar mit der Bildweite auf nur noch 20 Zeichen zurück, bleibt uns nur noch eine extrem verstümmelte Funktionstastenanzeige mit jeweils 3 Zeichen erhalten:

123	123	123	123	123
F1	F2	F3	F4	F5

Dies wirkt zwar unschön, läßt sich aber der Übersicht halber nicht anders ermöglichen (für die vollständige Anzeige der Funktionstastenbelegung hat Ihr MSX-Computer ja schließlich das Kommando KEY LIST!).

SCREEN 0/40 Zeichen: Anzeige von jeweils 7 Zeichen
SCREEN 0/37 Zeichen: Anzeige von jeweils 6 Zeichen
SCREEN 0/20 Zeichen: Anzeige von jeweils 3 Zeichen
SCREEN 1/32 Zeichen: Anzeige von jeweils 5 Zeichen
SCREEN 1/29 Zeichen: Anzeige von jeweils 5 Zeichen

Leider verfügen wir auf den Grafikbildschirmen über keine Funktionstastenanzeige. Obgleich es etwas schwierig und vor allem zeitintensiv ist, können wir auch hier selbstverständlich eine Anzeige der Funktionstastenbelegung erzielen: Dafür müssen wir die Ausgabe des Funktionstastenspeichers (je fünf Zeichen, da wir uns ja in einer 32- und nicht in einer 40-Zeichen-Darstellung befinden) in Zeile 24 erzwingen (gespeichert von &HF87F bis &HF91E)

Funktionstaste 1: &HF87F bis &HF88E
Funktionstaste 2: &HF88F bis &HF89E
Funktionstaste 3: &HF89F bis &HF8AE
Funktionstaste 4: &HF8AF bis &HF8BE
Funktionstaste 5: &HF8BF bis &HF8CE
Funktionstaste 6: &HF8CF bis &HF8DE
Funktionstaste 7: &HF8DF bis &HF8EE
Funktionstaste 8: &HF8EF bis &HF8FE
Funktionstaste 9: &HF8FF bis &HF90E
Funktionstaste 10: &HF80F bis &HF91E

und gleichzeitig die Abfrage der gedrückten bzw. nicht gedrückten SHIFT-Tasten abfragen, so daß wahrlich auch entsprechend die Funktionstasten 1 bis 5 oder 6 bis 10 zur Anzeige gebracht werden. Ich will an dieser Stelle gar nicht geauer auf den hier neu verwendeten Befehl (INP) eingehen, denn dazu gibt es andere Literatur. Hier soll halt nur die Abfrage der SHIFT-Tasten mit Hilfe von INP praktisch genutzt werden:

siehe Kurzlisting KEY ON 1 im Anhang

Noch vier Hinweise:

1) Sie können sich den Inhalt der aktuellen Funktionstastenbelegung (wie auch im Programm oben gerade geschehen) jederzeit aus den Speicherstellen &HF87F bis &HF91E herauslesen.

siehe Kurzlisting KEY ON 2 im Anhang

2) Die Standardfunktionstastenbelegung befindet sich im MSX-ROM an den Speicherstellen &H13A9 bis &H1448 Wollen Sie diese Belegung erzeugen, geben Sie bitte folgende zwei Befehle direkt (oder im Programm) in Ihren MSX-Computer ein:

```
DEFUSR=&H003E
A=USR(0)
```

Drücken Sie anschließend bitte eine der SHIFT-Tasten.

3) Wenn Sie in SCREEN 0 oder SCREEN 1 arbeiten, und die Funktionstastenausgabe angestellt ist (Anfangszustand oder KEY ON), können Sie Zeile 24 nur durch VPOKE ändern, wie folgendes Beispiel zeigt:

siehe Kurzlisting KEY ON 3 im Anhang

Der Urzustand (mit normaler Funktionstastenanzeige) läßt sich dann allerdings sehr einfach durch CLS, einen SCREEN-Befehl, das Drücken der CLS/HOME- bzw. einer SHIFT-Taste oder das Kommando KEY ON wiederherstellen.

4) Auch während des Programmablaufs wird der Wechsel der Funktionstastenanzeige nach dem Drücken der SHIFT-Taste angezeigt, jedoch nur zu dem Zeitpunkt, wenn eine Eingabe erfolgt:

```
A$=INKEY$
INPUT A
A$=INPUT$(1)
```

KEY ON 1

```
             -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 DRAW "B M 12,96"
50 PRINT #1,"Funktionstasten auf SCREEN
2"
60 PRINT #1,"    mit Wechsel bei SHIFT"
70 Z=0
80 FOR N=10 TO 210 STEP 50
90 DRAW "bm =n;,183"
100 Z=Z+1
110 A$=""
120 FOR M=1 TO 5
130 A$=A$+CHR$(PEEK(Z*16+&HF86F+M-1+(ZZ*
80)))
140 NEXT M
150 PRINT #1,A$;
160 NEXT N
170 A=INP(&HAA) AND &HF0
180 OUT &HAA,A OR 6
190 B=INP(&HA9)
200 IF B AND 1 THEN IF ZZ=1 THEN ZZ=0:LI
NE(10,183)-(255,191),15,BF
210 IF NOT B AND 1 THEN IF ZZ=0 THEN ZZ=
1:LINE(10,183)-(255,191),15,BF
220 GOTO 70
```

KEY ON 2

```
             -*-*-*-*-*-
10 SCREEN 0
20 COLOR 1,15
30 A$=STRING$(16,32)
40 FOR N=5033 TO 5192 STEP 16
50 FOR M=1 TO 16
60 IF PEEK(N-1+M)>31 THEN MID$(A$,M,1)=C
HR$(PEEK(N-1+M)) ELSE MID$(A$,M,1)=" "
70 NEXT M
80 PRINT A$
90 NEXT N
```

KEY ON 3

```
             -*-*-*-*-*-
10 KEY ON
20 SCREEN 0
30 COLOR 1,15
40 FOR N=920 TO 959
50 VPOKE N,INT(RND(1)*255)
60 NEXT N
```

Einstellung der Bildschirmweite auf den Textbildschirmen

siehe hierzu auch das Programm im Anhang: WIDTH 1
Programm 2

Sobald Sie Ihren MSX-Computer anstellen und das MSX-Logo verschwunden ist, befinden Sie sich auf SCREEN 0, der auf Bildschirmweite 37 eingestellt ist und über eine Funktionstastenanzeige verfügt.

Im Gegensatz zu den meisten anderen Homecomputern, bei denen Sie dafür wild im Speicher herumPOKEn müssen, können Sie bei Ihrem MSX-Computer für die Einstellung der Bildschirmweite (Darstellung von N Zeichen pro Zeile) auf einen extra BASIC-Befehl zurückgreifen:

WIDTH N

Für N können Sie jedoch nicht beliebige Werte einsetzen. Vielmehr gelten folgende Extremgrenzen:

SCREEN 0: WIDTH 1 bis 40
SCREEN 1: WIDTH 1 bis 32
SCREEN 2 und 3: WIDTH-Befehl nicht anwendbar bzw. die WIDTH-Eingabe bezieht sich dann automatisch auf SCREEN 0 bzw. SCREEN 1

Wie unterscheiden sich nun die Bildschirme SCREEN 0 und SCREEN 1 in der Textwiedergabe? Dafür muß man wissen, daß die MSX-Computer über eine hochauflösende Grafik von 256*192 Punkten verfügen. In diesem Rahmen müssen Sie bei SCREEN 0 folgende Beschränkung in Kauf nehmen:

SCREEN 0: 256/40 = 6 Punkte in der horizontalen Matrix

während bei SCREEN 1 (bis zu 32 Zeichen/Zeile) die volle Zeichenmatrix zu erkennen ist:

SCREEN 1: 256/32 = 8 Punkte in der horizontalen Matrix

d.h. in SCREEN 0 wird nicht das vollständige Characterzeichen dargestellt (8*8), sondern nur die ersten 6 Punkte der horizontalen 8-Punkte-Leiste. Damit dies nicht störend auffällt, haben die MSX-Konstrukteure ihren Zeichensatz so aufgebaut, daß immer nur die ersten 6 Punkte mit Inhalten besetzt sind. Denken Sie aber bitte immer an diese Beschränkung von SCREEN 0, bevor Sie sich nach einer gelungenen Grafikzeichendefinition die Haare raufen ob der unvollständigen Darstellung.

In SCREEN 1 ist das Schriftbild zwar ein bißchen gedehnt, dafür können Sie hier aber in jedem Fall von einer vollen Zeichendarstellung ausgehen (8*8). Aus diesem Grunde haben wir für unseren Characterzeichengenerator (siehe Programm 14 im Anhang) auch SCREEN 1 und nicht SCREEN 0 zur Bearbeitung herangezogen. Wenn Sie SCREEN 1 kurz nach dem Einstellen Ihres Computers ansteuern, werden Sie feststellen, daß auch hier im Anfangsstadium nicht die Höchstzahl 32 sondern ledigich 29 Zeichen pro Zeile dargestellt werden (entspricht WIDTH 29).

Warum nun sind im Anfangsstadium auf SCREEN 0 nur 37 Zeichen, auf SCREEN 1 gar nur 29 Zeichen darstellbar? Jeder Fernseher bzw. jedes einem internationalen Fernseher innewohnende Fernsehsystem (PAL, NTSC, SECAM) erfüllt unterschiedliche Normen, was die Bildschirmdarstellung anbetrifft. Ihrem MSX-Computer wurde ein in Deutschland üblicher Modulator (Bildübersetzer) eingebaut, der für die PAL-Norm ausgelegt ist. Ähnlich verfährt man in Amerika mit einem NTSC-Modulator und, da Ihr MSX-Computer ja

ein internationaler Computer ist, in anderen Ländern nach den entsprechend dort geltenden Sitten. Leider werden dadurch nicht die Unterschiede in der Bildschirmdarstellung vom Computer ausgemerzt. So erscheinen auf unserem PAL-System die mit dem MSX-Computer gezeichneten Kreise ellipsenförmig und ... bei 40-Zeichen-Darstellung auf SCREEN 0 und 32-Zeichen-Darstellung auf SCREEN 1 sind nicht mehr unbedingt alle Zeichen zu erkennen (oder zumindest sind einige Zeichen am Bildschirmrand nur teilweise sichtbar).

Um diesen internationalen Unterschieden pauschal Rechnung zu tragen, erzeugt jeder MSX-Computer beim Einschalten nicht die volle Zeilenbreite sondern nur einen Teil davon. Selbstverständlich können Sie mit WIDHT auf 40 Zeichen/Zeile bei SCREEN 0 bzw. 32 Zeichen/Zeile bei SCREEN 1 umschalten, aber sehen Sie selbst.

Da ein angeschlossener Monitor hingegen so verstellbar ist, wie Sie es wünschen (ein Nachrichtensprecher im Fernsehen würde bei solch einer Verstellaktion nicht mehr so aussehen wie erwartet), können Sie dort das Bild beliebig einstellen. Aus diesem Grund sind Monitoren mit Verstellreglern ausgestattet, die die zur Lösung unseres Problemfalles wichtigen Aufschriften tragen wie: Horizontal Width bzw. Vertical Width. Sind Sie stolzer Besitzer eines solchen Monitors, dürfte es Ihnen bei der entsprechenden Bildjustage anschließend kein Problem mehr bedeuten, was sowohl die vollständige Bilddarstellung als auch die Darstellung von wirklich runden Kreisen Ihres MSX-Computers anbetrifft.

In einem kurzen Programm will ich Ihnen noch einmal zeigen, daß auf SCREEN 0 im Gegensatz zu SCREEN 1 wahrlich nur eine 6*8-Matrix des 8*8-Grafikzeichens angezeigt wird:

siehe Kurzlisting WIDTH 1 im Anhang

Der WIDTH-Befehl verändert bei Eingabe einer neuen Bildschirmweite auch die in der letzten Zeile darstellbare Funktionstastenlatte. Während bei der Normaleinstellung von Funktionstaste 1 (COLOR 15,4,4) nach dem Einschalten (SCREEN 0, WIDTH 37) noch das Wörtchen COLOR sichtbar ist, bleibt bei Einstellungsänderung auf WIDTH 10 davon nur noch ein C als Orientierungshilfe übrig.

In diesem Fall ist es meist angebrachter, mit SCREEN ,,0 die Funktionstastenanzeige einfach auszuschalten und dadurch sogar noch eine 24. Bildschirmzeile hinzuzugewinnen.

Sie können die Bildschirmweite auch durch ein POKEn in die Speicherstellen &HF3AE, &HF3AF und &HF3B0 bewerkstelligen. Dabei bezieht sich Speicherstelle &HF3AE auf die Bildschirmweite in SCREEN 0, Speicherstelle &HF3AF auf die Bildschirmweite in SCREEN 1 und Speicherstelle &HF3B0 auf die Bildschirmweite im derzeit aktuellen SCREEN (die Eingaben in die Speicherstellen &HF3AE und &HF3AF zeigen erst zu dem Zeitpunkt Wirkung, wenn wir den entsprechenden SCREEN von neuem aufrufen). Selbstverständlich können Sie auch durch

```
PRINT PEEK(&HF3AE) bzw.
PRINT PEEK(&HF3AF)
```

die aktuelle Bildschirmweite von SCREEN 0 bzw. SCREEN 1 abfragen.

Noch ein paar Hinweise:

Anfangs wies ich bereits darauf hin, daß Sie nur im vorgegebenen Umfang (SCREEN 0 = 1 bis 40, SCREEN 1 = 1 bis 32) Bildschirmweiteänderungen vornehmen durchführen können. Beachten Sie dies nicht, wird Ihnen die Fehlermeldung 'Illegal function call' ausgegeben.

Mit Hilfe des BASE- und des VDP-Befehls können Sie mehrere Bildschirme nacheinander aus dem VRAM herausholen und darin blättern.

siehe Programm 2 im Anhang

Soweit Sie hier jedoch nur Bildschirme anzeigen wollen, gilt die Anweisung WIDTH nicht. Vielmehr müssen Sie diese Bildschirme direkt mit VPOKE ansprechen.

Bis zu einer WIDTH-Änderung merkt es sich Ihr MSX-Computer, welche aktuellen Werte zuletzt für SCREEN 0 bzw. SCREEN 1 bzgl. der Bildschirmweite eingegeben wurden, auch wenn Sie zwischendurch auf einen anderen SCREEN z.B. Grafikbildschirm SCREEN 2 umschalten und richtet sich nach der SCREEN-Umschaltung auch entsprechend danach.

Der Vollständigkeit halber sei vermerkt, daß SCREEN 2 über eine Konstante (nicht veränderbare Bildschirmweite) von 32 Zeichen verfügt (es sei denn Sie setzen die Buchstaben pixelweise anders nebeneinander, was durch die Grafikdarstellung selbstverständlich ohne weiteres möglich ist), während dies auf SCREEN 3 nicht lohnt, denn hier werden gar nur 8 Zeichen pro Zeile zur Darstellung gebracht (für Titel bestens geeignet).

```
WIDTH 1                 -*-*-*-*-*-
             10 COLOR 1,15
             20 SCREEN 0
             30 PRINT "Definition eines Vierecks"
             40 FOR N=1 TO 8
             50 READ A$
             60 VPOKE 2048+254*8+N-1,VAL("&B"+A$)
             70 NEXT N
             80 LOCATE 0,10
             90 PRINT "und so sieht es auf SCREEN 0 a
             us"
             100 PRINT
             110 PRINT CHR$(254)
             120 FOR N=1 TO 1000
             130 NEXT N
             140 SCREEN 1
             150 PRINT "Definition eines Vierecks"
             160 RESTORE
             170 FOR N=1 TO 8
             180 READ A$
             190 VPOKE 254*8+N-1,VAL("&B"+A$)
             200 NEXT N
             210 LOCATE 0,10
             220 PRINT "und so auf SCREEN 1"
             230 PRINT
             240 PRINT CHR$(254)
             250 GOTO 250
             260 DATA 11111111
             270 DATA 10000001
             280 DATA 10000001
             290 DATA 10000001
             300 DATA 10000001
             310 DATA 10000001
             320 DATA 10000001
             330 DATA 11111111
```

Befehle Grafik COLOR

Veränderung von Vordergrund-, Hintergrund- und Umrandungsfarbe

siehe hierzu auch die Programme im Anhang: COLOR 1
COLOR 2
COLOR 3
COLOR 4
CIRCLE 4

Alle MSX-Computer haben 15 Farben, die mit dem Befehl COLOR und Ziffern zwischen 0 und 15 für Vorder- und Hintergrund sowie Bldschirmumrandung angewählt werden können. Die zur Verfügung stehenden Farben sind:

0: durchsichtig (die Bildschirmumrandungsfarbe wird angezeigt; ist auch diese 0 = durchsichtig, kann man theoretisch tiefer liegende Zonen im Video-RAM anzeigen
1: schwarz
2: mittelgrün
3: hellgrün
4: dunkelblau
5: hellblau
6: dunkelrot
7: cyan
8: mittelrot
9: hellrot
10: dunkelgelb
11: hellgelb
12: dunkelgrün
13: magenta
14: grau
15: weiß

Da zu jeder Farbe ein entsprechend hellerer und/oder dunklerer Farbton vorhanden ist, kann man verhältnismäßig leicht dreidimensionale Körper auf dem Bildschirm darstellen, wie wir dies im folgenden Beispiel mit einem Würfel zeigen:

siehe Kurzlisting COLOR 1 im Anhang

Probieren Sie selbst einmal! Jedoch bitte besondere Vorsicht, wenn Sie mehr als zwei Farben gleichzeitig in einem 8*1-Feld darstellen wollen; es kommt bei Nichtbeachtung schnell zu ineinanderverlaufenden unschönen Farbdarstellungen.

siehe Kurzlisting CIRCLE 4 im Anhang

Der COLOR-Befehl besteht aus drei Parametern:

COLOR A,B,C

A=Vordergrundfarbe (Schreibfarbe)

B=Hintergrundfarbe (das Bildschirmpapier, auf welches wir mit der Vordergrundfarbe schreiben)

C=Umrandungsfarbe (soweit wir nicht in SCREEN 0 arbeiten erzeugen wir durch diesen Parameter, wenn er nicht gleich der Hintergrundfarbe ist, einen Bildschirmrahmen oben und unten)

Wenn wir den MSX-Computer anschalten, arbeiten wir mit der vorgegebenen Farbkombination 15,4,4 d.h. wir schreiben mit weißer Farbe=15 auf dunkelblauem Hintergrund=4 und verfügen ebenfalls über einen dunkelblauen Bildschirmahmen=4 (in SCREEN 0 nicht sichtbar). Diese Anfangsfarbkonstellation kann jederzeit durch Drücken der F6-Taste (Funktionstaste 1 mit SHIFT) wieder angewählt werden (Vorteil: weiß auf dunkelblau ist gut lesbar d.h. diese Farbkombination wirkt auf S/W- sowie Farbfernsehern sehr kontrastreich.

Wollen wir die Umrandungsfarbe (3.Parameter) ändern, jedoch Vorder- sowie Hintergrundfarbe beibehalten, können wir diese Parameter bei der Eingabe durch Kommata überspringen. Wünschen wir einen weißen Bildschirmrahmen, geben wir ein:

```
10 SCREEN 1
20 COLOR ,,15
```

Wir sprachen vorhin die Bedeutung der Funktionstaste 6 an (Farbzurückstellung auf den Anschaltzustand). Diese Notwendigkeit wird deutlich, wenn wir für Vorder- und Hintergrundfarbe die gleiche Farbe anwählen:

```
COLOR 15,15
```

Sie können machen, was Sie wollen - nichts ist mehr zu sehen. Drücken Sie einmal die F6-Taste und alles ist wieder in bester Ordnung (Sie sehen sogar die Tastendrücke, die Sie zwischendurch hilflos getätigt haben, denn COLOR 15,15 heißt nicht, daß der Bildschirm gelöscht wird, sondern lediglich, daß Vorder- und Hintergrundfarbe den gleichen Farbwert (=weiß) annehmen. Sie können somit bei gleicher Vorder- und Hintergrundfarbe ein gewünschtes Textbild durch ein Programm entwerfen und erst nach der Fertigstellung auf die passende Vorder- und Hintergrundfarbe umschalten. Der Effekt ist folgender: Der Bildschirm wird nicht mehr zeilenweise langsam vor dem Zuschauer aufgebaut, sondern ist mit einem Schlag fertig.

siehe Kurzlisting COLOR 2 im Anhang

Nun aber zur Unterscheidung des COLOR-Kommandos bei Anwendung in den einzelnen SCREENs und die sich daraus ergebenden Auswirkungen:

SCREEN 0: Hier kann man nur Vorder- und Bildhintergrund verändern. Die COLOR-Funktion wird sofort ausgeführt und bedarf nicht noch der Eingabe eines weiteren Kommandos (z.B. CLS). Es ist allerdngs auch nicht möglich, mehr als zwei Farben in SCREEN 0 gleichzeitig erscheinen zu lassen (dies wäre auch nur schwerlich möglich, denn wie wir beim Kommando BASE bereits erahren haben, gibt es bei SCREEN 0 auch keinen Farbtisch im Video-RAM). Bei der Farbumschaltung für Bildvorder- und Bildhintergrund werden nicht nur die neu eingegebenen Zeichen, sondern auch bereits bestehende Zeichen auf dem Bildschirm farblich verändert.

SCREEN 1: Da dieser SCREEN bereits über einen Bildschirmrahmen verfügt, können wir sämtliche drei Parameter des COLOR-Befehls hier anwenden (Vorder- und Hintergrund sowie Bildschirmrahmen). Sämtliche Parameter bewirken eine augenblickliche Farbänderung (wir brauchen nicht erst das MSX-BASIC-Kommando CLS einzugeben). Geben wir bei der Bildhintergrundfarbe 0 ein, schimmert der Bildschirmrahmen durch d.h. der Bildschirmrahmen wird somit zum Bildhintergrund. Der Befehl COLOR A,B,C hat keine Auswirkung darauf, wenn wir vorher Achtergruppen von Zeichen im Video-RAM unterschiedlich farblich gekennzeichnet haben (denn diese Informationen liegen ja im Color Table im VRAM); lediglich die weiteren Eingaben von farblich nicht umdefinierten Zeichen wird davon beeinflußt.

SCREEN 2 und SCREEN 3: Lediglich die Bildschirmumrandungsfarbe (3.Parameter) kann auch während des laufenden Programms geändert werden. Die Änderung der Vordergrundfarbe hingegen wird bei den Grafikbildschirmen 2 und 3 erst dann sichtbar, wenn angenommen, indem wir eine neue grafische Darstellung oder ein neues Textbild in unseren MSX-Computer eingeben. Bereits zuvor eingegebene Zeichen werden von der Änderung des ersten Parameters im COLOR-Befehl im Gegensatz zu der augenblicklichen Reaktion auf den Textbildschirmen.

Hingegen nehmen alle Grafikbefehle bis auf PRESET die aktuelle Vordergrundfarbe sofort an (PRESET nimmt die aktuelle Bildhintergrundfarbe an), es sei denn, wir geben dem Grafikbefehl selbst eine andere Farbe mit bei z.B.:

siehe Kurzlisting COLOR 3 im Anhang

Während in Zeile 30 die durch COLOR gewählte Farbe auch für die Grafik übernommen wird (Farbe 1), wird in Zeile 40 (größerer Kreis) die speziell angewählte Farbe (15=weiß) verwendet.

Wird die Farbe im Grafikbefehl direkt geändert, gilt dies nur für diese eine Grafik (anschließend wird wieder der zuletzt eingegebene COLOR-Wert verwendet, siehe Zeile 50). Wird hingegen mit COLOR eine andere Farbe eingegeben, bleibt diese Farbe erhalten, bis wir sie erneut ändern.

Noch ein Stück krasser sieht es aus, wenn wir Parameter Nummer 2 (Untergrundfarbe) durch den COLOR-Befehl ändern wollen. Die Farbänderung wird erst dann akzeptiert, wenn wir den Bildschirm nach Ausführung des COLOR-Befehls mit CLS oder SCREEN ... löschen (zur Unterscheidung der Auswirkung dieser zwei MSX-Befehle schauen Sie bitte unter CLS und SCREEN nach). Erst dann wird die neue Bildschirmuntergrundsfarbe auch auf den Grafikbildschirmen angezeigt.

Nun noch ein paar wichtige Tips und Tricks:

1) Für Maschinensprachprogrammierer wichtig: Wollen bzw. können Sie den COLOR-Befehl nicht verwenden, so ändern Sie die Vorder- und Hintergrundfarbe sowie die Bildschirmumrandung durch POKEn in folgende Speicherstellen:

```
Vordergrund: &HF3E9
Hintergrund: &HF3EA
Umrandung:   &HF3EB
```

Es ist auch möglich, die Vorder- und die Hintergrundfarbe (bei SCREEn 0) bzw. die Vorder- und die Umrandungsfarbe (bei SCREEN 1 bis 3) durch eine Direkteingabe in VDP-Register 7 zu ändern: Wollen Sie weiße Vordergrundfarbe und schwarze Hintergrund- (bei SCREEN 0) bzw. weiße Vordergrundfarbe und schwarze Umrandungsfarbe (bei SCREEN 1 bis 3), geben Sie einfach ein:

```
VDP(7)=&H1F
```

2) Die Taste CLS/HM bzw. der Befehl PRINT CHR$(12) haben auf den Grafikbildschirmen nicht die gleiche Bedeutung wie das MSX-BASIC-Kommando CLS zum Bildschirmlöschen (und damit zum eventuellen Verändern der Untergrundfarbe auf den Grafikbildschirmen).

siehe Kurzlisting COLOR 4 im Anhang

Die Taste CLS/HOME bzw. der Befehl PRINT CHR$(12) führen also ihre Bildschirmlöschfunktion nur auf den Textbildschirmen SCREEN 0 und SCREEN 1 durch.

COLOR 1

-*-*-*-*-*-

```
10 COLOR 15,1
20 SCREEN 2
30 IF Z=1 THEN C=1 ELSE C=3
40 DRAW "bm 112,96 c=c; r32 u32 l32 d32"
50 PAINT(140,86),3
60 IF Z=1 THEN C=1 ELSE C=12
70 DRAW "bm 144,96 c=c; e32 u32 n g32 l3
2 g32 r32 d32"
80 PAINT(154,76),12
90 PAINT(120,60),12
100 IF Z=1 THEN C=1 ELSE C=2
110 DRAW "bm112,96 c=c; r32 g16 l32 e16"
120 PAINT(116,103),2
130 IF Z=1 THEN GOTO 150 ELSE Z=1
140 GOTO 30
150 GOTO 150
```

COLOR 2

-*-*-*-*-*-

```
10 KEY OFF
20 COLOR 15,15
30 SCREEN 0
40 WIDTH 40
50 FOR N=1 TO 220
60 PRINT "MSX ";
70 NEXT N
80 COLOR 1,15
```

COLOR 3

-*-*-*-*-*-

```
10 COLOR 1,4
20 SCREEN 2
30 CIRCLE(128,96),20
40 CIRCLE(128,96),40,15
50 CIRCLE(128,96),60
60 GOTO 60
```

COLOR 4

-*-*-*-*-*-

```
10 COLOR 1,4
20 SCREEN 2
30 CIRCLE(128,96),20
40 PRINT CHR$(12)
50 GOTO 50
```

Befehle Grafik TAB

Ansteuerung von unterschiedlichem Zeichenabstand

siehe hierzu auch die Programme im Anhang: TAB 1
TAB 2
TAB 3

Sie kennen sicherich die Tabulator-Funktion, wie sie auf jeder Schreibmaschine zu finden ist: Dazu drücken Sie eine Taste, die den Druckwagen um eine bestimmte Anzahl von Zeichen nach rechts hin bewegt. Ist dieser Tabulatorabstand gar noch programmierbar, können Sie schließlich mit Ihrer Schreibmaschine eine hübsche, regelmäßig aufgebaute Tabelle erstellen.

Eine ähnliche Tabulatorsteuerung steht Ihnen auch bei Ihrem MSX-Computer zur Verfügung: Dazu drücken Sie bitte die mit den Buchstaben TAB oder aber einem Doppelpfeil nach rechts versehene Taste, und schon geht der Cursor um acht Stellen auf dem Bildschirm nach rechts.

Außerdem verfügt Ihr MSX-Computer aber in seinem BASIC über einen frei programmierbaren Tabulator, den Sie innerhalb von PRINT-Anweisungen mit der Buchstabenfole TAB direkt aufrufen können. Im Gegensatz zum einmaligen Tastendruck von vorhin müssen Sie den TAB-BASIC-Befehl aber noch zusätzlich mit einem entsprechenden Zahlwert versehen:

```
PRINT TAB(10);"MSX"
PRINT TAB(20);"MSX"
```

Die Ziffer, die hinter dem Wörtchen TAB angegeben ist, besagt, wie groß der Abstand von der letzten Cursorposition sein soll. Hierbei geht Ihr MSX-Computer immer von der ohne TAB zu erfolgenden nächsten Druckposition aus und rechnet die angegebenen Zeichen entsprechend hinzu. Wird allerdings als PRINT-Beginn nicht der linke Zeilenrand angesteuert, erfolgt die TAB-Ausgabe immer von der aktuellen linken Zeilenrandposition aus gerechnet (linker Zeilenrand = TAB(0)). Dazu drei Beispiele:

```
10 CLS
20 PRINT TAB(10);"MSX"
```

Der Ausdruck beginnt in Zeile 1 an der Cursorposition 10+1

```
10 CLS
20 LOCATE 10,0
30 PRINT TAB(15);"MSX"
```

Der Ausdruck beginnt in Zeile 1 an der Cursorposition 15+1

```
10 CLS
20 LOCATE 10,0
30 PRINT TAB(5);"MSX"
```

Der Ausdruck beginnt in Zeile 1 an der Cursorposition 10+1, da TAB(5) wegen seines kleineren Wertes im Vergleich zum vorstehenden LOCATE 10,0 ignoriert wird.

Hinweis: Wir haben in diesen drei Kurzprogrammen bei der Erläuterung jeweils zur Cursorposition '1' hinzugezählt, da die Zählweise der MSX-Computer bei 0 beginnt.

Sie können den Wert hinter der TAB-Funktion mit Zahlen zwischen 0 und 255 belegen, wobei das Vorpreschen des Cursors selbst über mehrere Zeilen kein Problem darstellt. Die Eingabe

```
PRINT TAB(0);"MSX"
```

würde bedeuten, daß das TAB-Kommando hier keine ergänzende Funktion zum PRINT-Befehl ausführt d.h. die Zeichenausgabe beginnt am linken Rand.

Haben Sie vorher die Bildschirmweite auf 40 Zeichen eingestellt (WIDTH 40), besagt die Befehlsfolge

```
PRINT TAB(40);"MSX"
```

daß die Ausgabe in der übernächsten Zeile beginnt

nächste Zeile: TAB(0) bis TAB(39)
übernächste Zeile: TAB(40) bis TAB(79)

Bedenken Sie aber bitte immer, daß Sie hinter den TAB-Befehl nur Zahlen zwischen 0 und 255 einsetzen können. Eine Eingabe größer als 255 bringt Ihnen eine 'Illegal function call'-Fehlermeldung.

Mit Hilfe des TAB-Kommandos erscheint es somit sehr einfach, einen formatierten Ausdruck zu erstellen. Schauen wir uns dies im folgenden Beispiel einmal an:

siehe Kurzlisting TAB 1 im Anhang

Auf den ersten Blick sieht die Bildschirmausgabe zwar formatiert aus, aber was ist geschehen, wenn wir uns die untereinander stehenden Zahlwerte einmal etwas genauer betrachten: Die Zehner der einen Zahl stehen direkt unter den Hundertern und Tausendern der anderen Zahlen, mathematisch gesehen unbrauchbar und auch die Übersicht, ob große oder kleine Zahl, läßt so stark zu wünschen übrig.

Wir können unser Programm erweitern, indem wir zum nützlichen TAB-Befehl auch noch die auszugebende Zahlengröße = das Format mit angeben. Dazu steht uns der Befehl PRINT USING zur Verfügung: Unsere ausgegebenen Zahlen liegen zwischen 1 und 1000, also ist die Darstellung in unserem Beispiel höchstens vierstellig; dafür ändern wir die entsprechende Zeile in unserem Programm bzw. erweitern sie durch PRINT USING "###":

siehe Kuzlisting TAB 2 im Anhang

Nun ist alles sauber dargestellt, wobei die Befehle TAB und PRINT USING Ihre Funktion in kooperativer Zusammenarbeit sauber erfüllen.

Noch ein paar Hinweise zur sinnvollen Anwendung des TAB-Befehls:

1. Sie könnnen TAB nicht nur auf dem Bildschirm in der Kombination mit PRINT verwenden, sondern ebenso auf einem angeschlossenen Drucker in Verbindung mit dem Druckkommando LPRINT.

2. Wollen Sie nach einer INPUT-Anweisung in der gleichen Zeile mit PRINT TAB weiterschreiben, müssen Sie den Cursor um eine Zeile nach oben hin versetzen. Dies tun Sie mit der Anweisung:

LOCATE gewünschter Abstand,CSRLIN-1

wie folgendes Programmbeispiel zeigt:

siehe Kurzlisting TAB 3 im Anhang

Wird die TAB-Anweisung so durchgeführt, daß bereits geschriebene Zeichen zu überqueren sind, so werden diese auf SCREEN 0 und SCREEN 1 gelöscht, auf den Grafikbildschirmen SCREEN 2 und SCREEN 3 bleiben sie jedoch erhalten.

TAB 1

```
            -*-*-*-*-*-
10 SCREEN Ø
20 COLOR 1,15
30 WIDTH 40
40 FOR N=1 TO 20
50 PRINT INT(RND(1)*2000);
60 PRINT TAB(10);INT(RND(1)*2000);
70 PRINT TAB(20);INT(RND(1)*2000);
80 PRINT TAB(30);INT(RND(1)*2000)
90 NEXT N
```

TAB 2

```
            -*-*-*-*-*-
10 SCREEN Ø
20 COLOR 1,15
30 WIDTH 40
40 FOR N=1 TO 20
50 PRINT USING "####";INT(RND(1)*2000);
60 PRINT TAB(10);USING "####";INT(RND(1)
*2000);
70 PRINT TAB(20);USING "####";INT(RND(1)
*2000);
80 PRINT TAB(30);USING "####";INT(RND(1)
*2000)
90 NEXT N
```

TAB 3

```
            -*-*-*-*-*-
10 SCREEN Ø
20 WIDTH 40
30 COLOR 1,15
40 PRINT "Bitte Zahlen kleiner als 100 e
ingeben"
50 PRINT
60 INPUT "Bitte eine Zahl ";A
70 IF A>99 THEN GOTO 60
80 LOCATE 25,CSRLIN-1
90 PRINT "Quadrat:";A*A
100 GOTO 60
```

Befehle Grafik LOCATE

Koordinatenansteuerung auf den Textbildschirmen

siehe hierzu auch die Programme im Anhang: LOCATE 1
LOCATE 2
LOCATE 3
LOCATE 4
LOCATE 5
LOCATE 6
LOCATE 7
LOCATE 8
LOCATE 9
LOCATE 10
LOCATE 11

Mit Hilfe des MSX-Kommandos LOCATE können wir zwei unterschiedliche Dinge erreichen:

1) Ansteuern einer bestimmten Bildschirmstelle auf den Textbildschirmen (SCREEN 0 und SCREEN 1)

2) Ein- und Ausschalten des Textcursors

zu 1: Um einen angenehm wirkenden Bildschirmaufbau zu erreichen, ist es nicht in jedem Fall sinnvoll, die Textausgabe an der äußersten linken Bildschirmposition zu beginnen, wie im folgenden Beispiel gezeigt:

siehe Kurzlisting LOCATE 1 im Anhang

Ein erster Schritt hin zur etwas gefälligeren Textdarstellung wäre das Einfügen von Leerzeichen bzw. Leerzeilen:

siehe Kurzlisting LOCATE 2 im Anhang

Dies ist aber auch nicht die Ideallösung, weil hierdurch nicht nur sehr viel Speicherplatz im Computer sinnlos vertan wird, sondern auch die Unübersichtlichkeit im Listing des Programms zunimmt. Man kann zwar mit Hilfe des TAB-Kommandos die horizontale Ebene besser aufteilen:

siehe Kurzlisting LOCATE 3 im Anhang

aber für das vertikale Freilassen von Zeilen müssen wir weiterhin PRINT-Leerzeilen einbauen. Auch nicht gerade effektiv, gerade wenn man eine Anzahl von Zeilen auf dem Bildschirm überspringen will.

Viel besser geht es mit Hilfe unseres LOCATE-Kommandos: Wir geben einfach die gewünschte Bildschirmposition an (ähnlich wie auf dem Grafikbildschirm mit dem Kommando DRAW "B M 128,96" zum Ansprechen des Bildschirmmittelpunkts), indem wir die gewünschte x- (Spaltenposition) und y- (Zeilenposition) Werte eingeben. Hierbei müssen wir uns allerdings bewußt sein, daß die Zählung bei Bildschirmposition 0,0 beginnt, was dem ersten Zeichen links oben auf dem Bildschirm entspricht.

Je nach vorheriger WIDTH-Einstellung (bei SCREEN 0 ist dies 1 bis 40, bei SCREEN 1 jedoch nur 1 bis 32) können somit die x- (Spaltenposition) Werte zwischen 0 und 39, die y- (Zeilenposition) Werte zwischen 0 und 31 liegen. Schauen wir uns dies einmal in einem Beispiel an:

siehe Kurzlisting LOCATE 4 im Anhang

Oder etwas übersichtlicher:

siehe Kurzlisting LOCATE 5 im Anhang

Im Gegensatz zum PRINT-Befehl ohne LOCATE (Eingabe von Leerzeichen bzw. Leerzeilen bzw. TAB-Kommandos), wobei uns bis zur angewählten Position der Bildschirm gelöscht wird (PRINT:PRINT:PRINT) hat der LOCATE-Befehl keine Auswirkung auf bereits bestehende Bildschirm-Zeilen und -Spalten. Dies zeigt das nächste Programm recht eindrucksvoll (das Semikolon muß allerdings hinter jedem PRINT-Befehl stehen, sonst führt Ihr MSX-Computer die Ausgabe samt Zeilenvorschub aus, was soviel bedeutet wie: lösche die Zeile hinter der Ausgabe bis zum Zeilenende):

siehe Kurzlisting LOCATE 6 im Anhang

In weiser Vorausahnung, Sie haben noch die Funktionstastenlatte engeschaltet, habe ich in diesem Programm den Darstellungsrahmen auf die Zeilen 0 bis 21 begrenzt. Hätte ich 22 zugelassen, so wäre es eventuell ab und zu doch zum Bildschirmscrolling gekommen (z.B. durch zufälliges Anwählen der Bldschirmposition x=38, y=21, d.h. die Ausgabe hätte in Zeile 22 fortgesetzt werden müssen, was soviel bedeutet wie: scrolle den Bildschirm um eine Zeile nach oben).

Ihr MSX-Computer denkt aber bereits so schlau, daß er eine begonnene Ausgabe (z.B. in Spalte 39) selbständig in der darauffolgenden Zeile fortsetzt ... sei es auch so, daß dafür ein Bildschirmscrolling einsetzt!

Es steht Ihnen frei, für x bzw. y beim LOCATE-Kommando Werte bis 255 einzusetzen. Verstehen tut Ihr MSX-Computer zwar auch die auf unseren Textbildschirmen nicht anwählbaren Positionen (z.B. x größer als 39), aber ausführen tut er in diesem Fall nur den extrem darstellbaren Wert (z.B. wenn x=100, so wird die Darstellung in Spalte x=39 bei SCREEN 0 und WIDTH 40 gebracht).

Wie bereits angeführt, können Sie den LOCATE-Befehl nur sinnvoll auf den zwei Textbildschirmen (SCREEN 0 und SCREEN 1) anwenden. Geben Sie einen LOCATE-Befehl ein, während Sie sich auf einer der Grafikseiten befinden, wird dieser Befehl nicht ausgeführt. Sie müssen auf dem Grafikbildschirm für ein Kommando wie LOCATE auf dem Textbildschirm den Befehl DRAW "B M x1,y1" verwenden).

zu 2: Manchmal ist es recht störend, immer wieder den Textcursorblock im laufenden Programm auf dem Bildschirm zu sehen. Indem wir im LOCATE-Befehl die x- und y-Ausgabe durch Kommata überspringen und im dritten Parameter eine 0 angeben, wird der Cursorblock gelöscht:

```
LOCATE ,,0
```

Dieser Effekt kann umgekehrt werden, indem wir an Stelle der 0 eine 1 oder einen Wert bis 255 eingeben:

```
LOCATE ,,1
```

Das "Auslöschen" des Cursorblocks hat folgenden Sinn: wir können auf diese Art und Weise selbst noch in der letzten Bildschirmzeile eine Ausgabe vornehmen, ohne daß der Bildschirm automatisch (und damit ungewollt) hochscrollt. Jedoch geben Sie darauf acht: Sie können den Cursor nur bei laufendem Programm ausstellen. Bereits zu dem Zeitpunkt, wenn Sie eine Eingabe vornehmen (INPUT, INPUT$ ...), erscheint der Cursor wieder auf dem Bildschirm, egal ob er zuvor mit LOCATE an- oder ausgestellt wurde.

Um dies zu zeigen, verbinden wir die beiden LOCATE-Funktionen einfach miteinander: die Bildschirmpositionsanwahl und das Löschen des Textbildschirmcursors:

siehe Kurzlisting LOCATE 7 im Anhang

oder mit sichtbarem Positionscursor:

siehe Kurzlisting LOCATE 8 im Anhang

Zudem können wir uns in Verbindung mit der Maschinenunterprogrammroutine (ON INTERVAL) auch wie bei anderen Computern einen blinkenden Cursor herstellen:

siehe Kurzlisting LOCATE 9 im Anhang

Übrigens:

1) Der Textbildschirmcursor entspricht im Computerzeichensatz CHR$(255). Wir können dies recht einleuchtend durch das folgende Programm demonstrieren:

siehe Kurzlisting LOCATE 10 im Anhang

Auf den ersten Blick ein dunkler Bildschirm. Nun gehen Sie aber einmal mit dem Cursor auf die einzelnen Buchstaben des zuletzt geschriebenen Textes. Augenblicklich ändert sich die gesamte Fläche auf dem Bildschirm so, wie der Cursor gerade aussieht (inverse Darstellung der unter ihm befindlichen Zeichen).

2) Das, was wir mit dem LOCATE-Befehl ansteuern können (x- und y-Koordinaten des Textcursors) können wir auch direkt durch POKEn in den Speicher nachvollziehen:

Speicherstelle &HF3DC = y-Position
Speicherstelle &HF3DD = x-Position

Also kommt die folgende LOCATE-Eingabe

```
LOCATE 15,10
```

folgendem POKEn in die zwei Speicherstellen gleich:

```
POKE &HF3DD,15
POKE &HF3DC,10
```

Sie können sich ganz allgemein merken: Jeder BASIC-Befehl läßt sich immer irgendwie durch POKE bzw. PEEK im Speicher nachvollziehen. Jedoch sollten wir so ehrlich sein und festhalten: Mit BASIC-Befehlen in Umgangsenglisch kommt man besser und schneller zurecht, als sich tausende von Speicherstellen zu merken!

Über die Speicherstellen (die sogenannten Systemvariablenadressen) können wir auch den Cursor ein- und ausstellen:

Cursor aus: POKE &HFCA9,0
Cursor an: POKE &HFCA9,1

3) Wie ich bereits an anderer Stelle gesagt habe, können wir mit LOCATE x,y jede beliebige Position auf dem Textbildschirm direkt anwählen und dorthin etwas schreiben. Es besteht auch die Möglichkeit, den Bildschirm von Position 0 bis Position 959 (bei WIDTH 40) durchzunummerieren. In diesem Fall müssen wir direkt im Videospeicher arbeiten, was folgendes Programm beweist:

siehe Kurzlisting LOCATE 11 im Anhang

In der Mitte des Bildschirms erscheint nun eine Senkrechte, die Zeile für Zeile (STEP 40) bis zum unteren Bildschirmrand führt.

LOCATE 1

```
            -*-*-*-*-*-
10 SCREEN Ø
20 COLOR 1,15
30 PRINT "Menue"
40 PRINT "1. Adressen eingeben"
50 PRINT "2. Adressen suchen"
60 PRINT "3. Adressen speichern"
```

LOCATE 2

```
            -*-*-*-*-*-
10 SCREEN Ø
20 COLOR 1,15
30 PRINT "      Menue
31 PRINT
40 PRINT "      1. Adressen eingeben"
50 PRINT "      2. Adressen suchen"
60 PRINT "      3. Adressen speichern"
```

LOCATE 3

```
            -*-*-*-*-*-
10 SCREEN Ø
20 COLOR 1,15
30 PRINT TAB(15) "Menue"
31 PRINT
40 PRINT TAB(10) "1. Adressen eingeben"
50 PRINT TAB(10) "2. Adressen suchen"
60 PRINT TAB(10) "3. Adressen speichern"
```

LOCATE 4

```
            -*-*-*-*-*-
10 SCREEN Ø
20 COLOR 1,15
30 LOCATE 15,5
40 PRINT "Menue"
50 LOCATE 10,7
60 PRINT "1. Adressen eingeben"
70 LOCATE 10,8
80 PRINT "2. Adressen suchen"
90 LOCATE 10,9
100 PRINT "3. Adressen speichern"
```

LOCATE 5

```
            -*-*-*-*-*-
10 SCREEN Ø
20 COLOR 1,15
30 LOCATE 15,5
40 PRINT "Menue"
50 FOR N=7 TO 9
60 LOCATE 10,N
70 READ A$
80 PRINT A$
90 NEXT N
100 DATA 1. Adressen eingeben,2. Adresse
n suchen,3. Adressen speichern
```

LOCATE 6

-*-*-*-*-*-

```
10 SCREEN Ø
20 COLOR 1,15
30 FOR N=1 TO 100
40 LOCATE INT(RND(1)*39),INT(RND(1)*21)
50 PRINT "MSX";
60 NEXT N
```

LOCATE 7

-*-*-*-*-*-

```
10 SCREEN Ø
20 COLOR 1,15
30 LOCATE Ø,Ø,Ø
40 PRINT "MSX"
50 GOTO 50
```

LOCATE 8

-*-*-*-*-*-

```
10 SCREEN Ø
20 COLOR 1,15
30 LOCATE Ø,Ø,1
40 PRINT "MSX"
50 GOTO 50
```

LOCATE 9

-*-*-*-*-*-

```
10 SCREEN Ø
20 COLOR 1,15
30 ON INTERVAL=10 GOSUB 60
40 INTERVAL ON
50 GOTO 50
60 IF Z=1 THEN Z=Ø ELSE Z=1
70 ON Z+1 GOTO 80,100
80 LOCATE ,,1
90 RETURN
100 LOCATE ,,Ø
110 RETURN
```

LOCATE 10

-*-*-*-*-*-

```
10 WIDTH 40
20 SCREEN Ø
30 COLOR 1,15
40 FOR N=1 TO 960
50 PRINT CHR$(255);
60 NEXT N
70 PRINT "MSX ist SUPER"
```

LOCATE 11

-*-*-*-*-*-

```
10 WIDTH 40
20 SCREEN Ø
30 COLOR 1,15
40 FOR N=20 TO 940 STEP 40
50 VPOKE N,33
60 NEXT N
```

Befehle Grafik POS

Abfrage der Spaltenposition auf den Textbildschirmen

siehe hierzu auch die Programme im Anhang: POS 1
POS 2
POS 3
POS 4
POS 5
POS 6

Es gibt mehrere Möglichkeiten, die Position des Cursors auf dem Textbildschirm zu bestimmen bzw. die aktuelle Position abzufragen. Einerseits können wir mit dem Befehl LOCATE (X,Y) angeben, an welcher X- und an welcher Y-Koordinate die nächste PRINT-Eingabe bzw. INPUT-Eingabe zu erfolgen hat. Mit dem Befehl CSRLIN (eigentlich ist dies eine Systemvariable, die lediglich auf ihren Wert abgefragt wird) können wir zudem erfahren, in welcher Linie=Bildschirmzeile wir uns zur Zeit mit unserem aktuellen Cursor auf dem Textbildschirm befinden.

Während also mit CSRLIN die Y-Koordinate des Cursors abgefragt wird, haben wir entsprechend zur Abfrage der Zeichenposition in der Zeile den Befehl POS (oder besser: die Systemvariable POS), die uns Auskunft darüber gibt, an welcher X-Koordinate sich unser Cursor gerade befindet.

Probieren wir einmal aus, wie die Ausgabe der POS-Systemvariablen vor sich geht:

siehe Kurzlisting POS 1 im Anhang

Wenn Sie das Programm mit RUN starten, wird innerhalb weniger Sekunden der Bildschirm mit dem Wort "MSX" vollgeschrieben. Hinter jedem "MSX" steht eine Zahl ausgedruckt, die die aktuelle Zeichenposition angibt. Wollen wir auch noch erfahren, in welcher Zeile wir uns gerade befinden, fragen wir zusätzlich auch noch mit CSRLIN die betreffende Zeilennummer ab:

```
45 PRINT CSRLIN;
```

Wozu, so werden Sie sich sicherlich fragen, kann man die Ausgabe der X-Y-Koordinaten auf dem Textbildschirm mit POS und CSRLIN sinnvoll nutzen? Ich versuche es, dies Ihnen an zwei praktischen Beispielen aufzuzeigen:

siehe Kurzlisting POS 2 im Anhang

Um einen allgemeinen Ursprung für alle MSX-Anwender zu schaffen, muß bei Verwendung des POS-Befehls festgelegt sein, wie viele Zeichen pro Zeile darstellbar sind bzw. auf welchem SCREEN gearbeitet wird (WIDTH 40 und SCREEN 0). Der vorgegebene Text in Zeile 50 reicht bis zur Saltenposition 27. Ist Ihre Namenseingabe nun länger als der noch verügbare freie Zeichenraum, wird durch PRINT ein Zeilenvorschub erzeugt und daraufhin in der nächsten Bildschirmzeile mit der Ausgabe Ihres Namens fortgefahren. Also können Sie den POS(...)-Befehl sinnvoll dazu einsetzen, wenn Sie bei variabler Eingabe doch noch eine saubere Ausgabe auf dem Bildschirm erzielen wollen (denn es gibt, wie in unserem Beispiel lange Namen wie 'Rainer-Maria Rilke' und auch kurze Namen wie 'Hans Sachs'; für beide Ausgaben würde unser kleines Programm mit Hilfe des POS-Kommandos die passende Ausgabe erzeugen.

Eine weitere Anwendungsmöglichkeit von POS bezieht sich auf den formatierten Bildschirmausdruck:

siehe Kurzlisting POS 3 im Anhang

In diesem Fall wird nur dann das Wort 'MSX' ausgegeben, wenn eine bestimmte Bildschirmspalte erreicht wird. Ähnliches ließe sich anwenden auf einen formatierten Ausdruck eines Rechenblattes oder Kalkulationssheets. Jedoch sollten Sie dazu noch einen weiteren Befehl miteinbeziehen: PRINT USING erlaubt es Ihnen, in einer vorgewählten Größenordnung rechtsbündig Zahlen auszugeben, während allein unter Hinzuziehung des POS-Befehls jeweils die erste Ziffer der auszugebenden Zahl linksbündig = unmathematisch auf dem Bildschirm erscheinen würde:

siehe Kurzlisting POS 4 im Anhang

und die Kopplung mit PRINT USING sieht so (sauberer) aus:

siehe Kurzlisting POS 5 im Anhang

Während es bei CSRLIN ausreicht, diese Variable einfach mit PRINT auf dem Bildschirm auszugeben, braucht hingegen der POS-Befehl bzw. die POS-Variable noch ein zusätzliches Argument. Hierbei ist es egal, welche Zahl wir einsetzen, das Ergebnis bleibt immer gleich, nur - es muß ein Argument so vorliegen, daß der POS-Befehl ausschaut wie: POS(N).

Vorhin wies ich kurz darauf an, daß wir an Stelle des Gebrauchs von Kommandos, die den Werten von Systemvariablen entsprechen (wie bei POS(N) und CSRLIN) auch direkt den Speicher befragen können:

der aktualle Wert von POS(N) an Speicherstelle &HF3DD
der aktuelle Wert von CSRLIN an Speicherstelle &HF3DC

Hierzu ein Hinweis: Die Zählweise von POS(N) und CSRLIN beginnt jeweils bei 0, während die Angabe in den obigen zwei Speicherstellen des RAM mit 1 anfängt.

Gibt es für die Grafikbildschirme auch einen Befehl, um den Standort des aktuellen Cursors abzufragen? Nein, es gibt dazu keinen besonderen Befehl. Wir können anstatt dessen den Speicher abfragen und zwar gleich nach vier verschiedenen Werten:

der absolute Wert der X-Koordinate an Speicherstelle &HFCB3
der absolute Wert der Y-Koordinate an Speicherstelle &HFCB5
der relative Wert der X-Koordinate an Speicherstelle &HFCB7
der relative Wert der Y-Koordinate an Speicherstelle &HFCB9

siehe Kurzlisting POS 6 im Anhang

Hierzu zwei Zahlbeispiele:

1. Zeile 30: DRAW "N R8"

&HFCB3: 136 &HFCB5: 96 &HFCB7: 128 &HFCB9: 96

2. Zeile 30: CIRCLE(128,96),20

&HFCB3: 20 &HFCB5: 96 &HFCB7: 128 &HFCB9: 96

Die Unterscheidung absolut und relativ ergibt sich daraus, daß viele Grafikbefehle von einem bestimmten Bildschirmpunkt ausgehen, während die Zeichnung an einer anderen Bildschirmstelle relativ zum Ausgangspunkt verschoben dargestellt wird (z.B. Eingabe des Kreismittelpunkts, während die Ausgabe im Radiusabstand stattfindet; ist der Kreis fertig gezeichnet, kehrt der Grafikcursor vom Kreiszeichnen zum absoluten Punkt (Kreismittelpunkt) wieder zurück.).

Der LOCATE-X-Y-Befehl auf dem Textbildschirm wird auf den Grafikbildschirm entweder durch DRAW "B M X,Y" oder durch PRESET(X,Y) ersetzt.

Noch zwei Hinweise: Bei vielen Grafikbefehlen ist es möglich, auch außerhalb des sichtbaren Bildschirmfensters Koordinaten anzusprechen. Wird hier der POS(N)-Befehl abgefragt, wird als Wert -1 ausgegeben.

Sie können auf den Grafikbildschirmen den Befehl POS(N) zwar nicht direkt anwenden, dafür gibt er Ihnen aber auch dort noch die Information über die Spaltenposition des Cursors auf dem zuletzt angewählten Textbildschirm (sei es nun SCREEN 0 oder SCREEN 1).

```
POS 1                    -*-*-*-*-*-
               10 SCREEN 0
               20 COLOR 1,15
               30 PRINT "MSX";
               40 PRINT POS(1);
               50 GOTO 30
```

POS 2

```
              -*-*-*-*-*-
10 SCREEN Ø
20 COLOR 1,15
30 WIDTH 40
40 INPUT "Bitte Ihren Namen eingeben
         ";A$
50 PRINT "Es freut mich ganz besonders,
dass ich  begruessen kann: Herrn/Frau ";
60 IF LEN(A$)>40-POS(Ø) THEN PRINT:PRINT
 A$ ELSE PRINT A$
```

POS 3

```
              -*-*-*-*-*-
10 SCREEN Ø
20 COLOR 1,15
30 WIDTH 40
40 FOR N=1 TO 1000
50 IF POS(1)/4 =INT(POS(1)/4) THEN PRINT
 "MSX ";
60 NEXT N
```

POS 4

```
              -*-*-*-*-*-
10 SCREEN Ø
20 COLOR 1,15
30 WIDTH 40
40 FOR N=1 TO 1000
50 IF POS(1)/5 =INT(POS(1)/5) THEN PRINT
 INT(RND(1)*999);ELSE PRINT "  ";
60 NEXT N
```

POS 5

```
              -*-*-*-*-*-
10 SCREEN Ø
20 COLOR 1,15
30 WIDTH 40
40 FOR N=1 TO 1000
50 IF POS(1)/5 =INT(POS(1)/5) THEN PRINT
 USING "###";INT(RND(1)*999);ELSE PRINT
"  ";
60 NEXT N
```

POS 6

```
              -*-*-*-*-*-
10 SCREEN 2
20 PSET(128,96)
30 DRAW"N R8
40 A=PEEK(&HFCB3)
50 B=PEEK(&HFCB5)
60 C=PEEK(&HFCB7)
70 D=PEEK(&HFCB9)
80 SCREENØ
90 PRINT A;B;C;D
100 PRINT
110 LIST
```

Grafik Befehle LPOS

Abfrage der aktuellen Position des Druckkopfes

siehe hierzu auch Programm im Anhang: LPOS 1

Mit diesem Befehl (bzw. der Abfrage dieser Variablen) können Sie erfahren, an welcher Position (Spalte) sich der Druckkopf des angeschlossenen Druckers gerade befindet.

Dies ist besonders dann wichtig, wenn Sie zum Beispiel Tabellen auf dem Drucker ausgeben wollen und verlangen, daß alle ausgegebenen Daten direkt untereinander stehen (wie es nun einmal bei einer tabellarischen Ausgabe auch so sein soll).

Das folgende Programm führt diese Funktion gleichzeitig auf dem Bildschirm und dem Drucker aus:

siehe Kurzlisting LPOS 1 im Anhang

Sowohl auf dem Bildschirm (POS(1)) als auch auf dem Drucker (LPOS(1)) werden dreimal 2 Zahlen ausgegeben, die tabellarisch untereinander stehen.

Eine andere Anwendung des LPOS(1)-Befehls ist die, bei einem ausgegebenen Text den Drucker daraufhin zu hinterfragen, ob noch genügend Platz in der ausgewählten Zeile für das folgende Wort vorhanden ist, andernfalls muß mit einem LPRINT-Befehl ein Zeilenvorschub veranlaßt werden.

Übrigens:

1) Wir können die LPOS(1) (gleich Line Print X POSition) auch abfragen, indem wir direkt im MSX-Systemvariablenspeicher folgende Funktion durchführen:

```
PRINT &HF415
```

Durch die direkte Arbeit mit der Speicherstelle &HF415 können wir dem LPOS(1)-Parameter auch einen Wert zuweisen und somit eine Art von LOCATE-Befehl durchführen:

```
POKE &HF415,40
```

d.h.: der Printerkopf befindet sich nun an Position 40.

2) Sinnvoller und besser als den LPOS(1)-Befehl (bzw. die Variable im Speicher) zur Druckkopfsteuerung zu gebrauchen, ist es fast immer, die Ausgabe durch LPRINT TAB, den luxuriösen LPRINT USING-Befehl oder gar das zusammengesetzte Kommando LPRINT USING TAB zu steuern (Beispiele siehe unter TAB).

Anders verhält es sich mit der Kontrolle des Druckkopfes, damit nicht über das Zeilenende bis in die nächste Zeile die Ausgabe ungewollt fortgesetzt wird.

LPOS 1

-*-*-*-*-*-

```
10 FOR N=1 TO 3
20 READ A,B
30 IF POS(1)<>5 THEN PRINT " ";:GOTO 30
40 PRINT A;
50 IF LPOS(1)<>5 THEN LPRINT " ";:GOTO 5
0
60 LPRINT A;
70 IF POS(1)<>20 THEN PRINT " ";:GOTO 70
80 PRINT B;
90 IF LPOS(1)<>20 THEN LPRINT " ";:GOTO
90
100 LPRINT B;
110 PRINT
120 LPRINT
130 NEXT N
140 DATA 10,20,30,40,50,60
```

Befehle Grafik CSRLIN

Abfrage der Zeilenposition des Cursors auf den Textbildschirmen

siehe hierzu auch das Programm im Anhang: CSRLIN 1

Mit Hilfe dieses Befehls können Sie erfahren, in welcher Bildschirmzeile sich der Textcursor gerade befindet (anwendbar nur in SCREEN 0 und SCREEN 1).

Dadurch können Sie es unterbinden, daß unbeabsichtigt ein Bildschirmabschnitt überschrieben wird, den Sie vorher mühsam erstellt haben. Wir demonstrieren dies an einem Beispiel: in Bildschirmzeile 10 schreiben wir den Begriff 'MSX'. Nun fordern wir unseren MSX-Computer dazu auf, den gesamten Bildschirm mit dem Wort 'SUPER' zu beschreiben, es sei denn, er stößt auf Bildschirmzeile 10 (obgleich auf dem Bildschirm Zeile 10 durch Nachzählen deutlich zu erkennen ist, akzeptiert der MSX-Computer dieselbige anstatt dessen als Zeile 9, denn die Zählweise beim Computer ist nicht 1, 2, 3 ... 10 sondern 0, 1, 2 ... 9):

siehe Kurzlisting CSRLIN 1 im Anhang

Unser MSX-Computer hat also auf dem Bildschirm nachgeschaut, in welcher Zeile er sich befindet; erst mit der Sicherheit, hier nichts bereits Bestehendes zu überschreiben (IF CSRLIN 9), führt er sein geplantes Vorgehen auch aus (THEN PRINT "SUPER").

Sie können den CSRLIN-Befehl auf den Textbildschirmen (SCREEN 0 und SCREEN 1) lediglich im Rahmen von 0 bis 22 (wenn die Funktionstastenanzeige in Zeile 23 erscheint) bzw. von 0 bis 23 (wenn keine Funktionstastenanzeige angestellt ist) gebrauchen. Ist der Text einmal aus dem Bildschirmrahmen herausgescrollt, ist er unwiderbringlich verloren.

Sinnvoll verwenden läßt sich der CSRLIN-Befehl erst, wenn wir ihn z.B. in einem selbsterstellten Texteingabeprogramm verwenden. Hierzu müssen wir dann allerdings neben der Y-Position (durch CSRLIN) auch die X-Position des Cursors auf dem Bildschirm feststellen können (durch POS) sowie den Cursor nach eigenem Gutdünken irgendwohin auf dem Bildschirm positionieren können (durch LOCATE).

Übrigens: Wir können die Y-Position des Cursors (Bildschirmzeile, die durch CSRLIN abgefragt wird) auch im Speicher nachlesen und zwar in Speicherstelle &HF3DD.

Wollen wir die Y-Koordinate des Cursors auch auf dem Grafikbildschirm erfahren, können wir dazu nur eine weitere Speicherstelle befragen und zwar &HFCB5.

Es existiert für die Abfrage des Y-Grafikcursors auch noch Speicherstelle &HFCB9. In letztgenannter Speicherstelle erfahren wir schließlich, wie weit sich der Grafikcursor bei einer relativen Adressierung (durch STEP z.B. bei den Befehlen CIRCLE oder LINE) bereits vom Ursprungspunkt entfernt hat.

CSRLIN 1

-*-*-*-*-*-

```
10 SCREEN 0
20 COLOR 1,15
30 LOCATE 0,9
40 PRINT "MSX"
50 FOR M=1 TO 1000
60 NEXT M
70 LOCATE 0,0
80 FOR N=0 TO 20
90 IF CSRLIN<>9 THEN PRINT "SUPER" ELSE
LOCATE 0,CSRLIN+1
100 NEXT N
```

Befehle Grafik CHR$

Direkte Ansteuerung des gesamten Zeichensatzes (0 bis 255)

siehe hierzu auch die Programme im Anhang: CHR$ 1
CHR$ 2
CHR$ 3
CHR$ 4
CHR$ 5

Wenn Sie bisher die alphabetische Reihenfolge beim Lesen eingehalten haben, ist Ihnen der Befehl CHR$ bereits aus der Beschreibung des ASC-Befehls ein Begriff: Während der ASC-Befehl jedes darstellbare Zeichen in eine Dezimalzahl umwandelt (nicht mit dem Zeichen sondern genau mit dieser Zahl arbeitet intern auch Ihr MSX-Computer), passiert bei Anwendung des CHR$-Befehls genau das Entgegengesetzte. Sie geben ein:

```
PRINT CHR$(65)

Ergebnis: A
```

Auf diese Art und Weise können wir uns mit Hilfe des folgenden Programms einmal den gesamten (darstellbaren) Zeichensatz der MSX-Computer ansehen:

siehe Kurzlisting CHR$ 1 im Anhang

Erst erscheinen einige Textzeichen (Punkt, Komma ...), dann die Ziffern 0 bis 9, anschließend die großen sowie die kleinen Buchstaben, schließlich noch die internationalen Sonderzeichen (ä, ö ...) und ganz zum Schluß ein paar Mathematik- und ein paar Grafikzeichen. Sie werden sich sicherlich fragen, warum wir denn bei der Darstellung des Zeichensatzes auf dem Bildschirm (siehe CHR$ 1) nicht mit der Nummer 1 bzw. gar der Nummer 0 begonnen haben. Das hat folgenden Grund: die ersten 32 Charakterzeichen sind nicht ohne weiteres lesbar darstellbar, zumindest was die Bildschirmanzeige anbetrifft. Probieren Sie dazu einmal folgendes aus:

```
PRINT CHR$(7)
```

Soweit Sie Ihren Computer hören (sprich: der Lautsprecher des an Ihren MSX-Computer angeschlossenen Fernsehers Töne von sich gibt), erklingt ein BEEP-Ton (siehe auch unter BEEP) und auf dem Bildschirm erscheint die alt vertraute Buchstabenkombination: Ok. Somit erfolgt z.B. bei CHR$(7) keine unmittelbar sichtbare Anzeige auf dem Bildschirm.

Oder wie wäre es mit einem:

```
PRINT CHR$(13)
```

Auch hier tut sich auf den ersten Blick nicht viel (anscheinend). Nur eine Leerzeile wird zusätzlich auf dem Bildschirm ausegeben.

Bleiben wir gleich bei diesen zwei Charakterzeichen und zeigen, wie sinnvoll doch ihre Verwendung im Einzelfall sein kann: Geben Sie dazu folgendes Programm in Ihren MSX-Rechner ein:

siehe Kurzlisting CHR$ 2 im Anhang

Wenn Sie dieses Programm mit RUN starten und der Anweisung Folge leisten, passiert nichts Weltbewegendes. Geben Sie allerdings eine Zahl ein, die entgegen der gestellten Forderung von Zeile 30 größer als 100 ist, BEEPt Ihr MSX-Computer kurz und macht Sie somit auf die Fehleingabe einerseits durch den Satz, der auf dem Bildschirm erscheint ('Bitte befolgen Sie meine Anweisungen.') und das FehlerBEEPen (beides Zeile 40) darauf aufmerksam.

CHR$(13) hingegen simuliert das Drücken der ENTER-Taste. Lassen wir dies praktisch erproben, indem wir die Funktionstasten 1 und 2 folgendermaßen neu belegen:

```
KEY 1,"LIST"
KEY 2,"LIST"+CHR$(13)
```

Falls Sie noch das Programm von vorhin im Speicher haben, probieren Sie nun Funktionstaste 1 und Funktionstaste 2 aus: Nach dem Drücken von Funktionstaste 1 müssen Sie erst noch die ENTER-Taste drücken (damit der Befehl auch ausgeführt wird), während bei Drücken der Funktionstaste 2 dies automatisch (halt durch das hinzugefügte CHR$(13)) geschieht.

Bei einem anderen BASIC-Befehl kann auf die Verwendung des CHR$-Kommandos ebenfalls nicht verzichtet werden: Mit der Variablenzuweisung SPRITE$ wird das Aussehen von Sprites festgelegt. Sind wir anspruchslos und uns reicht ein einstreifiges Spritemuster, so können wir dies mit der Tastatur direkt eingeben:

```
SPRITE$(0)="UUUUUUUU"
```

Soll dieses Spritemuster aber z.B. in der Mitte durch zwei Leerzeilen (keine Punkte gesetzt) aufgelockert werden, müßten wir mit der Tastatur CHR$(0) eingeben; gar nicht so einfach. Anstatt der schwierigen Direkteingabe über die Tastatur verwenden wir einfach unseren CHR$-Befehl:

```
SPRITE$(0)="UUU"+CHR$(0)+CHR$(0)+"UUU"
```

Noch ganz andere Zeichenfolgen lasse sich von der Tastatur aus eingeben, aber in einem Programm gar nicht so leicht abfragen. Insbesondere beim CP/M-Betriebssystem, das auf den MSX-Rechnern mit angeschlossener Diskettenstation zum Laufen gebracht werden kann, ist die Verwendung von Kombinationen der Taste CTRL mit einer Buchstabentaste sehr beliebt. Wie können/sollen wir jedoch in einem selbstgeschriebenen BASIC-Programm abfragen, ob jemand eine Tastenkombination mit CTRL gedrückt hat? Dafür gibt es eine sehr einfache Programm Formel mit Hilfe des in diesem Kapitel behandelten BASIC-Befehls CHR$:

siehe Kurzlisting CHR$ 3 im Anhang

Solange Sie in diesem Prgramm keine Taste zusammen mit der CTRL-Taste drücken, passiert gar nichts. Erst wenn Sie z.B. CTRL-E eingeben (bis zum Ende der Zeile löschen), zeigt der Computer in der Bildschirmanzeige an, daß er Sie richtig verstanden hat.

Weiterhin können Sie bei manchen Druckern mit Hilfe des CHR$-Befehls verschiedene Schriftbilder anwählen. Zumindest bei EPSON-Druckern beginnt die Druckeransteuerung mit dem ESC-Kommando, das als CHR$-Zeichen der Zahl 27 entspricht. Wollen Sie zum Beispiel verstärkten Druck einstellen (Emphasized mode), geben Sie folgendes Druckersteuerzeichen ein:

```
PRINT CHR$(27);"E"
```

Schließlich eine letzte Anwendung des CHR$-Befehls (es gibt noch unendlich viele mehr, aber die dürften Sie sich nach dem intensiven Durcharbeiten dieses Kapitels selbst entwickeln können): im Programm wird eine Eingabe in Großschrift verlangt, aber nicht immer ist man sicher, ob dies auch wahrlich befolgt wird (es dürfte Ihnen ja bereits bekannt sein, daß MSX-Computer wohlweislich zwischen den kleinen und den großen Buchstaben unterscheiden können). Wandeln wir also einfach die Kleinschrifteingabe in Großschrift um, wenn sie uns so nicht gefällt:

siehe Kurzlisting CHR$ 4 im Anhang

Geben Sie ein Wort Ihrer Wahl ein. Ob nun in Groß- oder in Kleinbuchstaben (oder beides), auf dem Bildschirm erfolgt in jedem Fall anschließend die Ausgabe in Großbuchstben.

Noch zwei Hinweise: Es dürfte sich von selbst erklären, daß auch hier wieder die magische Computerzahl 255 zu beachten ist. Um keine Fehlermeldung (Illegal function call) zu riskieren, darf die Zahl beim Befehl CHR$ in jedem Fall nur zwischen 0 und 255 liegen.

CHR$(255) ist gleichbedeutend mit dem Cursorzeichen. Sie können durch die Anzeige dieses Zeichens einen netten Effekt erzeugen:

siehe Kurzlisting CHR$ 5 im Anhang

Da der Cursor viele Male auf dem Bildschirm angezeigt wird, geschieht nun auch die Cursorzeichenveränderung gleichzeitig überall dort, wo das Cursorzeichen abgebildet wird; daher der rasche Bildschirminhaltwechsel, wenn er im Endprodukt auch nur mit der schnellwechselnden Darstellung lediglich eines Characterzeichens zusammenhängt.

CHR$ 1

```
         -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN Ø
30 WIDTH 40
40 FOR N=32 TO 255
50 PRINT CHR$(N);
60 NEXT N
```

CHR$ 2

```
         -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN Ø
30 WIDTH 40:INPUT "Bitte eine Zahl
                 kleiner als 100 einge
ben ";A
40 IF A>99 OR A<Ø THEN PRINT "Bitte folg
en Sie meinen Anweisungen";CHR$(7)
50 GOTO 30
```

CHR$ 3

```
         -*-*-*-*-*-
10 CLS
20 A$=INKEY$
30 IF A$="" THEN GOTO 20
40 IF ASC(A$)<32 THEN GOTO 50 ELSE GOTO
20
50 PRINT "Sie haben CTRL-";CHR$(ASC(A$)+
64);" gedrueckt"
60 GOTO 20
```

CHR$ 4

```
         -*-*-*-*-*-
10 SCREEN Ø
20 COLOR 1,15
30 WIDTH 40
40 Z=96
50 INPUT "Bitte geben Sie ein Wort ein "
;A$
60 FOR N=1 TO LEN(A$)
70 IF MID$(A$,N,1)>="a"THEN MID$(A$,N,1)
=CHR$(ASC(MID$(A$,N,1))-32)
80 NEXT N
90 PRINT A$
```

CHR$ 5

```
         -*-*-*-*-*-
10 SCREEN Ø
20 COLOR 1,15
30 WIDTH 40
40 FOR N=1 TO 23
50 PRINT STRING$(40,255);
60 NEXT N
80 PRINT "Bitte fuehren Sie den Cursor m
it den    Pfeiltasten auf Buchstaben"
```

Befehle Grafik ASC

Umwandlung von Characterzeichen in entsprechende Zahlwerte

siehe hierzu auch die Programme im Anhang: ASC 1
ASC 2
ASC 3

Wie Sie bereits wissen, stellt Ihr MSX-Computer intern alle Zeichen (ob Buchstaben, Ziffern oder Grafik) mit den Zahlen '0 und 1' bzw. den elektrischen Zuständen 'aus und an' dar.

Der Befehl ASC hilft uns dabei, die mit der Tastatur eingegebenen Zeichen auf dem Bildschirm als Dezimalzahlen darzustellen:

```
10 PRINT ASC("A")
```

Ergebnis: 65

Da Ihr MSX-Computer das A in diesem Fall nicht als Variable oder Schublade ansieht, müssen Anführungszeichen links und rechts vom Buchstaben A gesetzt werden (um es als String zu kennzeichnen).

Wir können aber auch direkt eine Stringvariable vom Computer berechnen lassen. Besteht diese nur aus einem Zeichen, läuft der Vorgang genauso wie oben im Beispiel dargestellt ab. Bei einem längeren String interpretiert Ihr MSX-Computer lediglich das erste Zeichen:

```
10 A$ = "Haus"
20 PRINT ASC(A$)
```

Ergebnis: 72

Wollen wir uns die ursprüngliche Darstellung des Buchstaben H (in Nullen und Einsen) auf dem Bildschirm ansehen, wählen wir eine etwas aufwendigere Darstellungsform, die aber mit Hilfe des luxuriösen MSX-BASIC-Wortschatzes gar nicht so schwierig zu meistern ist:

```
10 A$ = "Haus"
20 PRINT BIN$(ASC(A$))
```

Ergebnis: 1000100

Nun zur Anwendung des ASC-Befehls in unseren Programmen:

1) Wir wollen Tastatureingaben erkennen, die auf dem Bildschirm nicht direkt dargestellt werden können.

2) Wir wollen Grafikzeichen auf einem Nicht-MSX-Drucker für jederman lesbar ausgeben.

3) Wir wollen ein Zeichen, das mit Hilfe der Tastatur direkt angesteuert werden kann, in seinem Aussehen verändern.

zu 1): Manchmal ist es sinnvoll, in einem Programm Warteschleifen einzubauen und den Benutzer mit einem Satz auf dem Bildschirm dazu aufzufordern, z.B. die Eingabetaste (gekennzeichnet mit ENTER oder RETURN) zur Programmfortsetzung zu drücken. Das Problem bei Verwendung des INPUT$-Kommandos ist, daß wir die Eingabe auf dem Bildschirm nicht direkt lesen können:

siehe Kurzlisting ASC 1 im Anhang

Ergebnis: eine Leerzeile

Auf dem Bildschirm erscheint nichts, wenn nach der geforderten Eingabe die ENTER-Taste gedrückt haben. Schauen wir uns aber die Länge des Strings mit dem Namen A$ an, sehen wir, daß doch, obgleich ohne Anzeige auf dem Bildschirm, ein Zeichen eingegeben wurde:

```
PRINT LEN(A$)
```

Ergebnis: 1

Verwenden wir nun zur genaueren Interpretation des eingegebenen Zeichens den Befehl ASC:

siehe Kurzlisting ASC 1a im Anhang

Ergebnis: 13

Die Taste ENTER hat also für den Computer den Wert 13. Nun können wir unser Programm so verändern, daß erst dann mit dem Programmablauf fortgefahren wird, wenn die ENTER-Taste gedrückt wurde:

siehe Kurzlisting ASC 1b im Anhang

zu 2): Wie Sie wissen, hat das MSX-System einen vom normalen leicht abgewandelten Zeichensatz. Zwar ist die Darstellung der Zahlen, Buchstaben und Zeichen zwischen 32 (Leerzeichen) und 126 (Schlangenlinie) genormt, von 1 bis 31 und von 127 bis 254 werden aber spezielle Zeichen des MSX-Systems dargestellt.

Mit Hilfe eines MSX-Druckers können diese Zeichen auch samt und sonders direkt durch eine LPRINT-Ausgabe dargestellt werden; anders sieht es aber aus, wenn wir einen anderen Drucker an die CENTRONICS = Druckerschnittstelle anschließen. Zwar wird auch jeder andere Drucker etwas ausgeben, aber bei Verwendung von Zeichen außerhalb des genormten Bereichs (32 bis 126) wird es zu Überraschungen (Umstellung auf eine andere Schriftart, Umstellung auf hochauflösende Grafikdarstellung usw.) kommen. Am besten, Sie probieren einmal aus, ob Ihr Drucker ein MSX-Drucker oder ein normaler Drucker ist:

```
10 SCREEN ,,,,0
20 LPRINT " "     'GRAPH-Taste und P drücken
```

Erscheint nun auf dem Drucker das eingegebene Blockzeichen, ist alles in Ordnung und Sie können beruhigt unter 'zu 3)' weiterlesen, denn Sie sind somit Besitzer eines MSX-Druckers und brauchen sich nicht mit den Problemen der Nicht-MSX-Druckerbesitzer auseinanderzusetzen. Ist dies allerdings nicht der Fall, müssen Sie zukünftig bei jedem Programm, das Grafik auf dem Drucker ausgeben soll, folgenden Befehl zuvor eingeben:

```
SCREEN ,,,,1
```

Dadurch wird dem Computer gesagt, daß er sämtliche Grafikzeichen auf dem angeschlossenen Drucker als Leerzeichen ausgeben soll.

Wie können Sie aber trotzdem für Ihren Freund, der auch einen MSX-Computer besitzt, oder einer Zeitschrift, der Sie Ihre selbsterstellten MSX-Programme für den Ausdruck zusenden wollen, die Grafikzeichen mitteilen? Entweder Sie geben beim Ausdruck anstelle der Grafikzeichen Leerzeichen ein und betätigen sich anschließend als kleiner Künstler (Einmalen der Grafikzeichen in das Listing), oder Sie wandeln die auf Ihrem Drucker nicht darstellbare Grafik in die entsprechenden Zahlzeichen um, denn jedes Grafikzeichen ist ja letztendlich nur eine Zahl (s.o.). Dafür müssen Sie jedes Grafikzeichen entweder aus einer Tabelle entnehmen (siehe Übersicht im Anhang) oder das gewünschte Grafikzeichen mit Hilfe des ASC-Befehls entschlüsseln. Mit Hilfe des folgenden kurzen Programms kann dies gleich für bis zu 255 aufeinanderfolgende Zeichen erfolgen:

siehe Kurzlisting ASC 2 im Anhang

Anstelle der Grafikzeichen (die der Nicht-MSX-Drucker ja nicht darstellen kann) können Sie nun in Ihr Listing eingeben:

```
LPRINT CHR$(180);CHR$(...);...
```

Wie Sie unter dem Befehl CHR$(nachlesen können, werden mit diesem Kommando die Zahlen wieder in Buchstaben, Ziffern oder Grafikzeichen umngewandelt.

zu 3): Wir müssen die ASC-Nummer des zu verändernden Zeichens kennen, damit wir die richtige Adresse im VRAM (Video-RAMspeicher) verändern können. Z.B. wollen wir das Leerzeichen in seinem Aussehen verändern. Die Errechnung der Speicherstelle, wo die Struktur des Leerzeichens beginnt, finden wir mit folgendem kurzen Programm:

siehe Kurzlisting ASC 3 im Anhang

Ergebnis: 2304

Hinweis: Auf die Befehle BASE und SCREEN wird an anderer Stelle in diesem Buch genauer eingegangen.

Die Zahl 2304 besagt uns, daß an dieser Speicherstelle im VRAM die Abspeicherung des Leerzeichens beginnt (insgesamt sind es acht Speicherstellen, die für das Aussehen des Leerzeichens zuständig sind: 2304 bis 2311). Geben wir in die Speicherstelle einen anderen Zahlwert ein und Sie werden merken, was passiert:

```
VPOKE 2304,255
```

Mit einem Mal hat sich anscheinend der ganze Bildschirm verändert. Deutlicher wird dies noch, wenn wir CLS eingeben. Die plötzliche Umpolung des Bildschirms hängt aber nur damit zusammen, daß unser Bildschirm fast nur aus Leerzeichen besteht. Ihr MSX-Computer wandelt also bei Veränderung des Zeichenaussehens - in unserem Fall des Leerzeichens - alle gleichartigen Zeichen auf dem Bildschirm (mit der gleichen ASC-Nummer) im Aussehen um, sogar im Listing! Rückgängig können wir diesen Vorgang aber auch wieder machen, sei es durch die Eingabe eines SCREEN-Befehls (alle Veränderungen im VRAM werden auf Standardwerte zurückgesetzt) oder durch die Eingabe eines weiteren POKEs:

```
VPOKE 2304,0
```

Nun ist alles wieder in bester Ordnung, wobei Sie hoffentlich einiges über den ASC-Befehl hinzugelernt haben.

Noch zwei Hinweise zum nützlichen Gebrauch des ASC-Befehls:

1) Die Ausgabegröße bewegt sich auch bei der Entschlüsselung des Zeichenaussehens mit ASC bis zum Grenzwert der magischen Computerzahl 255. Sie werden also immer Zahlen zwischen 0 und 255 vom Computer ausgegeben bekommen, nie größere und auch nie kleinere Zahlwerte.

2) Hüten Sie sich davor, den ASC-Befehl bei leeren Strings anzuwenden z.B.:

```
10 A$=""
20 PRINT ASC(A$)
```

Aus nichts kann Ihr MSX-Computer auch nichts errechnen. Das Ergebnis ist deshalb eine Fehlermeldung: 'Illegal function call'.

ASC 1

```
                    -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 0
30 PRINT "Bitte ENTER-Taste druecken"
40 A$=INPUT$(1)
50 PRINT A$
```

ASC 1a

```
                    -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 0
30 PRINT "Bitte ENTER-Taste druecken"
40 A$=INPUT$(1)
50 PRINT ASC(A$)
```

ASC 1b

```
                    -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 0
30 PRINT "Bitte ENTER-Taste druecken"
40 A$=INPUT$(1)
50 IF ASC(A$)=13 THEN PRINT "Fertig" ELS
E GOTO 30
```

ASC 2

```
                    -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 0
30 INPUT "Bitte etwas eingeben ";A$
40 FOR N=1 TO LEN(A$)
50 PRINT MID$(A$,N,1);" = ";ASC(MID$(A$,
N,1)),
60 NEXT N
```

ASC 3

```
                    -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 0
30 PRINT "Leerzeichen = CHR$(32)"
40 START=2048+32*8
50 PRINT START
```

Befehle Grafik DRAW
Programmiersprache zum Zeichnen von Linien

siehe hierzu auch die Programme im Anhang: DRAW 1
DRAW 2
DRAW 3
DRAW 4
DRAW 5
DRAW 6
DRAW 7
DRAW 8
DRAW 9
DRAW 10
DRAW 11
DRAW 12
DRAW 13
DRAW 14

Das Kommando DRAW unter dem Titel 'Befehl' zu behandeln, kommt einer Untertreibung gleich. Übertrieben wäre es hingegen, bei Verwendung des DRAW-Befehls davon zu sprechen, daß dies neben BASIC die zweite Programmiersprache der MSX-Computer wäre. Zwar stecken hinter dem DRAW-Befehl, wie nicht zuletzt die Länge dieses Kapitels beweist, eine Vielzahl von Unterkommandos. Diese Unterkommandos dienen lediglich einem Zweck: dem Zeichnen von Linien. Trotzdem kommt dieses Befehlsspektrum einer echten Grafikprogrammiersprache wie LOGO nur um Bruchteile nahe (lediglich das LOGO-Unterprogramm Schildkrötengrafik ist mit dem Befehl DRAW der MSX-Computer nachvollziehbar).

Nach dieser Untertreibung der Fähigkeiten des DRAW-Befehls (der Hersteller des MSX-BASIC, die amerikanische Softwarefirma MICROSOFT, spricht beim Befehl DRAW von der GML = Graphic Macro Language) gehen wir ran in medias res und schauen uns diesen Befehl mal etwas genauer an:

siehe Kurzlisting DRAW 1 im Anhang

Bevor wir auf die in diesem Programm verwendeten DRAW-Befehle genauer zu sprechen kommen, sei darauf hingewiesen, daß es sich bei der Verwendung des DRAW-Befehls um ein Tätigsein in der Grafikebene handelt. Grafik läßt sich allerdings nur auf SCREEN 2 und SCREEN 3 abbilden, andernfalls gibt es die Fehlermeldung: Illegal function call.

Gehen wir die Zeichen hinter dem DRAW-Befehl Teil für Teil durch: Die Anführungszeichen verraten uns, daß es sich hier um einen String handelt (ähnlich wie A$="Hans"). Im Augenblick ist das noch nicht wichtig, aber warten Sie bitte einen Augenblick ab, dann werden Sie die Möglichkeiten der Stringbehandlung bei DRAW noch zu schätzen lernen.

Zu Beginn teilen wir unserem MSX-Computer mit, daß er mit seinem Grafikcursor zum Bildschirmmittelpunkt (128, 96) gehen soll. Mit 'BM 128,96' erreichen wir genau dies; das 'B' bedeutet ausgeschrieben BLIND, was so viel heißt wie: ziehe aber keine Linie auf dem Bildschirm, bis du Koordinate 128,96 erreicht hast, während das 'M' ausgeschrieben eigentlich heißen würde: MOVE d.h. bewege den Grafikcursor von seiner letzten Position aus zu der nachstehenden Koordinate hin (128, 96).

Ausgeschrieben würde 'B M 128,96' also heißen: BLIND MOVE TO 128,96 (Bewegung des Cursors ohne Zeichnung zur Koordinate 128,96).

Haben Sie selbst noch mäßige Erinnerungen an das Erlernen der englischen Sprache in der Schule, wird Ihnen das Übersetzen der vier Richtungen nicht gerade sehr schwer fallen:

nach oben = Up
nach unten = Down
nach links = Left
nach rechts = Right

Da wir Zeit und Speicherplatz bei der Eingabe sparen wollen, reicht es unserem MSX-Computer, wenn wir nur den Amnfangsbuchstaben der entsprechenden Richtung eingeben und zwar

'U' für Up
'D' für Down
'L' für Left
'R' für Right

Zu jeder Richtungsangabe braucht unser MSX-Computer nur noch eine Zahl, um zu wissen, wie weit die Reise denn gehen soll. Dabei besagt z.B. die Eingabe 'R8' schlicht und einfach: gehe 8 Bildschirmpunkte nach rechts.

Obgleich wir nun erst ein paar der DRAW-Unterbefehle kennengelernt haben, können wir nun unser Beispiel deuten:

DRAW "B M 128,96 U8 R8 U8 R8 D16 L16"

Gehe an Koordinatenposition 128,96, ohne bis dahin eine Linie zu zeichnen; bewege den Cursor 8 Punkte aufwärts, dann 8 Punkte nach rechts, noch einmal 8 Punkte aufwärts, wiederum 8 Punkte nach rechts und von dort aus 16 Punkte nach unten und 16 Punkte nach links, fertig ist unsere kleine Grafik, eine Treppe.

Zur Schreibweise hier nur so viel: Damit Sie alles besser lesen können, haben wir in diesem Buch Leerzeichen zwischen die einzelnen DRAW-Unterbefehle gesetzt. Wenn Sie wollen, können Sie darauf gerne verzichten; Sie können auch Semikolons an jeder Stelle dort einsetzen, wo wir ein Leerzeichen zur Abtrennung gebraucht haben. Nur an einer Stelle muß ein Komma zur Abtrennung eingegeben werden: Wenn eine Bildschirmkoordinate (X,Y) mit zwei Zahlen angesprochen wird, muß zwischen diesen Zahlen zur Abtrennung ein Komma gesetzt werden.

Gehen wir weiter in unseren DRAW-Unterbefehlen: Wir brauchen uns nicht nur mit den vier Himmelsrichtungen zufrieden zu geben, wir können auch Zwischenwerte verwenden (in der Wetterkunde spricht man dann z.B. von SO oder Südosten bzw. NW oder Nordwesten ...). Da auch diese Richtungen im MSX-BASIC der Einfachheit halber nur

aus einem Buchstaben bestehen, haben die Entwickler dieser DRAW-Unterbefehle eine blendende Idee gehabt: Gehen wir das Alphabet vom Buchstaben 'E' angefangen der Reihe nach durch: Die Zwischenhimmelsrichtungen heißen somit einfach:

Nord-Ost = E
Süd-Ost = F
Süd-West = G
Nord-West = H

Mit Hilfe unserer acht Himmelsrichtungen können wir nun auch ein Haus malen (kennen Sie das noch aus der Kinderzeit? Ein Haus zeichnen, ohne den Stift abzusetzen und ohne eine Linie zweimal zu zeichnen?):

Übersetzen wir uns eine mögliche Reihenfolge beim Zeichnen in den DRAW-Befehl:

siehe Kurzlisting DRAW 2 im Anhang

Aber in unserem MSX-Kommando DRAW steckt noch sehr viel mehr drin! Wie wäre es zum Beispiel, wenn wir das dargestellte Häuschen etwas größer oder gar kleiner darstellen wollen? Ganz einfach, werden Sie sagen: Soll das Häuschen doppelt so groß erscheinen, verdoppeln wir ganz einfach die Zahlangaben hinter dem DRAW-Befehl, so daß schließlich in Zeile 30 steht:

```
30 DRAW "B M 128,96 R16 U16 L16 E8 F8 G16 U16 F16"
```

So klappt es wohl auch, ist aber laut der Meinung der Softwareingenieure von MICROSOFT noch zu aufwendig. Anstatt dessen verwenden wir einen weiteren Buchstaben mit Zahlangabe, der den Vergrößerungs- bzw. Verkleinerungsfaktors unserer Zeichnung angibt. Der hierzu verwendete Buchstabe ist das "S", was so viel bedeutet wie Skalenfaktor. Wollen wir in normaler Größe zeichnen (d.h. DRAW "R8", was soviel bedeutet wie: bewege den Grafikcursor um acht Punkte nach rechts), müssen wir Skalenfaktor vier ("S4") anwählen. Für eine doppelt so große Darstellung (Größenverhältnis = 2:1) muß der S-Wert zweimal 4 = 8 betragen (DRAW "S8"). Soll auf die Hälfte verkleinert werden (Größenverhältnis

= 1:2) muß der S-Wert 2 betragen (die Hälfte des Ausgangswerts = 4/2 = 2). Wir können dieses Vergrößerungs- bzw. Verkleinerungsspielchen mit Werten von 1 bis 255 und dem Buchstaben "S" weiterführen. Dabei ist es unserem MSX-Computer egal, ob wir über die Bildschirmbegrenzung hinausschießen (was leicht einmal vorkommen kann, wenn wir einen zwei- oder gar dreistelligen Skalenfaktor benutzen). In diesem Fall wird die extrem außerhalb des Bildschirmrahmens liegende Zeichnung am Bildschirmrand angezeigt, wie z.B. in:

siehe Kurzlisting DRAW 3 im Anhang

Unterbrechen Sie bitte die Bildschirmdarstellung durch Drücken der Tasten CTRL und STOP und starten Sie es von neuem mit RUN. Zu Ihrem großen Erstaunen wird nun unser Quadrat gar nicht mehr abgebildet, anstatt dessen nur unser übergroßes Quadrat. Wie ist das nun zu erklären? Bei der Beschreibung des COLOR-Befehls hatte ich bereits darauf hingewiesen: der MSX-Computer behält vieles in Erinnerung, so z.B. die einmal mit COLOR angewählte Vorder-, Hintergrund- und Rahmenfarbe. Ähnlich verhält es sich auch mit dem Skalenfaktor: wird er nicht rechtzeitig wieder zurückgesetzt (in unserem Fall auf Normalgröße = "S4"), bleibt er auch beim Einsatz weiterer DRAW-Befehle erhalten. Dies hat den Vorteil, daß man den Skalenfaktor für oftmaliges Benutzen nicht in jeder Zeile neu definieren muß! Also müßte Zeile 30 in unserem Programm ein klein wenig geändert werden, wollen wir es immer wieder von neuem in seinem Urzustand ablaufen lassen:

```
30 DRAW "B M 128,96 S4 R8 U8 L8 D8"
```

Einen weiteren Buchstaben, den wir in den DRAW-Befehl einsetzen können, ist das A = wie Angel = Winkel. Hiermit ist es möglich, unsere erstellten Zeichen zu drehen und zwar im 90-Grad-Rhythmus:

siehe Kurzlisting DRAW 4 im Anhang

Hier ist noch nichts gedreht worden, damit Sie sich merken können, wie unsere Zeichnung im Urzustand ausgesehen hat: eine Treppenstufe, die vom Mittelpunkt her aufwärts führt. Nun verändern wir Zeile 30 mit unserem neuen A-Kommando:

```
30 DRAW "B M 128,96 A1 R8 U8 R8"
```

Wie auch immer man unsere Zeichnung nun deuten will: die Treppenstufe führt abwärts: das Rechts-Kommando wurde ebenso wie das Aufwärts-Kommando um 90 Grad im Uhrzeigersinn gedreht, so daß nun aus der Funktion R (nach rechts) eine Funktion D (nach unten) wurde; ebenso verhält es sich mit der U(aufwärts)-Funktion: Sie wurde auch um 90 Grad im Uhrzeigersinn gedreht, so daß aus aufwärts ein rechts wurde. Also können wir folgende Regel aufstellen:

```
A = 1 d.h. U = R  (aufwärts = rechts)
           R = D  (rechts = abwärts)
+90 Grad   D = L  (abwärts = links)
           L = U  (links = aufwärts)
```

Hätten wir einen Turm gezeichnet, so würde derselbige jetzt auf der ursprünglich rechten Seite flachliegen.

Geben wir höhere Werte für das Kommando A im DRAW-Befehl ein, so vergrößert sich der Winkel immer mehr (im Uhrzeigersinn), jedoch immer nur in 90 Grad-Schritten:

```
A = 2  d.h.  U = D  (aufwärts = abwärts)
             R = L  (rechts = links)
+180 Grad    D = U  (abwärts = aufwärts)
             L = R  (links = rechts)

A = 3  d.h.  U = L  (aufwärts = links)
             R = U  (rechts = aufwärts)
+270 Grad    D = R  (abwärts = rechts)
             L = D  (links = abwärts)
```

Mit Hilfe eines kleinen Unterprogramms ist es gar möglich, daß wir unsere Zeichnungen mit dem DRAW-Befehl auch um jeden beliebigen anderen Winkel drehen.

Wozu brauchen wir denn diesen DRAW-A-Befehl? Haben Sie es gelernt, damit umzugehen, wäre es nun z.B. möglich, sehr einfach einen kleinen Kompaß zu zeichnen. Dafür brauchen Sie nur einmal die Pfeilform zu definieren, und dieselbige dann durch einen vierfach umdefinierten Richtungsfaktor mehrmals zu zeichnen:

siehe Kurzlisting DRAW 5 im Anhang

Obgleich immer von neuem unser Pfeil mit seiner Pfeilspitze nach oben (wie in Zeile 30 angegeben) gezeichnet werden soll, beeinflußt der immer größer werdende A-Faktor im DRAW-Befehl die Richtung letztendlich. Auf diese Art und Weise wird die Richtung verdreht und ein Pfeil nach oben (Norden), einer nach rechts (Osten), einer nach unten (Süden) und einer nach links (Westen) entsteht und hilft uns leicht, das gewünschte Kompaßaussehen zu erstellen. Denken Sie einmal nach, um wie vieles ausführlicher das Programm geworden wäre, wenn jedesmal die Richtung hätte umgedreht (und damit auch umgerechnet) werden müssen.

Wenn wir zu weiteren Unterkommandos des DRAW-Befehls gelangen, wird Ihnen deutlich werden, wie einfach und luxuriös es doch im Endprodukt ist, mit dem Kommando DRAW zu arbeiten, wenn man weiß, wie.

Noch ein Tip zur sinnvollen Anwendung des DRAW-A-Kommandos: Später im Buch haben wir ein Listing abgebildet, das uns die Verwendung von Buchstaben und Ziffern auf den Grafikbildschirmen unter Zuhilfenahme des DRAW-Befehls gestattet: mit diesem DRAW-Buchstaben ist es dann selbstverständlich auch möglich, die Schrift an einer gezeichneten Gerade (Vertikale) entsprechend hochlaufen zu lassen oder gar ein Detail in unserer Zeichnung auf dem Kopf stehend zu betiteln.

Bevor Sie sich dieses eindrucksvolle Programm eintippen, hier noch ein weiteres DRAW-Unterkommando: mit Hilfe des Buchstabens N beauftragen wir unseren MSX-Computer, nach Beendigung seines Zeichenauftrags an den Ursprungspunkt zurückzukehren.

Um diesen Zeichenbefehl praktisch demonstrieren zu können, vereinfachen wir unseren vorhin gezeichneten Kompaß, indem wir die Pfeilspitze weglassen und das seinerzeit so langes Programm ist zu etwas mehr als einem Einzeiler geschrumpft:

siehe Kurzlisting DRAW 6 im Anhang

Zu Beginn des DRAW-Befehls bekommt der MSX-Computer den Auftrag, bei Bildschirmposition 128,96 (Bildschirmmittelpunkt) mit der Zeichnung zu beginnen. Der anschließend eingegebene Buchstabe N besagt, daß der folgende Zeichenbefehl zwar ausgeführt werden soll, daß aber hiernach wieder zuletzt angesprochenen Grafikpunkt (in unserem Fall der Mittelpunkt bei 128,96) zurückgekehrt werden soll.

Wie wir im vorigen Kapitel (A-DRAW-Unterbefehl) gelernt haben, besagt die steigende Angabe für A, daß der Befehl U (aufwärts) nacheinander als gedreht um 90, 180 und schließlich 270 Grad verstanden wird. So entsteht letztendlich unsere Windrose.

Wollen Sie es größer haben? Das dürfte, wenn Sie bisher zum DRAW-Kommando alles aufmerksam gelesen haben, kein Problem mehr sein: Setzen Sie ein 'S' für den Skalenfaktor vor die eigentliche Zeichenroutine und geben anschließend eine Zahl größer als 4 ein zum Beispiel:

```
30 DRAW"B M 128,96 S12 N A0 U8 N A1 U8 N A2 U8 N A3 U8"
```

Bereits zu Beginn dieses Kapitels haben wir die DRAW-Unterbefehle M und B angesprochen: Mit M und zwei darauf folgenden Koordinaten (die durch ein Komma voneinander getrennt sein müssen) geben wir an, daß der Grafikcursor sich von der letzten Bildschirmposition dorthin bewegen sollen (M = to move = bewegen), allerdings, indem er in der vorher eingestellten Farbe die Linie zeichnen soll.

Wünschen wir eine Bewegung des Cursors ohne Zeichnung zum angegebenen Zielpunkt, setzen wir vor das M noch ein B, was so viel bedeutet wie: gehe blind zum Zielpunkt = ziehe keine Linie.

Da bei Verwendung von DRAW "B M" nur der Grafikcursor bewegt, nicht aber gezeichnet wird, kann dieser Befehl gut dazu verwendet werden, ihn ähnlich dem LOCATE-Befehl auf den Textbildschirmen zur punktweisen Positionierung (auch von Grafiktexten) zu gebrauchen.

Wir können aber noch anders mit den Befehlen M und B umgehen: Falls Sie in den MSX-Kommandos LINE, PSET und CIRCLE firm sind, so ist Ihnen sicherlich auch bereits der Unterschied zwischen relativer (mit STEP) und absoluter Adressierung bekannt: Wir müssen bei der absoluten Adressierung davon ausgehen, daß unser Bildschirm mit der Position 0,0 in der linken oberen Bildschirmecke beginnt (wollen wir in der Mitte des Bildschirms etwas zeichnen, müssen wir uns im Rahmen der zur Verfügung stehenden Bildpunkte (256 * 192) den Mittelpunkt ausrechnen (128,96) und geben für diese Stelle den Befehl, mit der Zeichnung zu beginnen). Einfach gesagt: absolut heißt, 0,0 ist oben links, 255,191 ist unten rechts und 128,96 ist der Bildschirmmittelpunkt. Bei der relativen Adressierung hingegen bestimmen wir uns einen neuen Anfangspunkt, von wo aus (STEP) nun so und so viele Schritte nach rechts, oben, links oder unten gegangen werden sollen. Diese relative Adressierung hat den Vorteil, daß wir die bereits erstellten Zeichnungen auch an ganz anderen Bildschirmstellen wieder auftragen können.

Während wir bei den Hauptgrafikbefehlen einfach STEP für die relative Adressierung eingeben können, geht dies innerhalb unseres DRAW-Kommandos einfacher vonstatten: Wir verwenden bei der Koordinateneingabe wie gehabt die Zahlwerte, geben aber zwecks relativer Adressierung ein Pluszeichen (+) oder ein Minuszeichen (-) zusätzlich ein. Liest dies unser MSX-Computer, weiß er sofort, daß der ursprüngliche Punkt nun als neuer Nullpunkt anzusehen ist. Als Beispiel wollen wir ganz einfach vier Kästchen nebeneinander zeichnen:

siehe Kurzlisting DRAW 7 im Anhang

Einfach und einleuchtend, nicht wahr? Verändern wir Zeile 50 so, daß eine Diagonale aus den Kästchen zusammengesetzt wird und zur Abbildung gelangt:

```
50 DRAW "B M +8,+8 R4 U4 L4 D4"
```

Wenn Sie die relative Adressierung erst einmal bei viel komplizierteren Zeichenstrukturen verwenden, werden Sie auch diese Feinheit des MSX-DRAW-Grafikkommandos zu schätzen lernen (z.B. arbeitet unser bereits vorhin angesprochener DRAW-Zeichensatz auf dem Grafikbildschirm nur so und nicht anders, denn sonst könnten wir ja unsere Schrift nur absolut an einer bestimmten Bildschirmposition erscheinen lassen und nicht die Zeichen in gleichem Abstand nebeneinander screiben.

Ein anderes praktisches Anwendungsbeispiel wäre die Darstellung von Schachfiguren auf einem Schachbrett: 16 schön geformte Bauern heißt es hier zu zeichen ... nichts einfacher, als unter Hinzuziehung der relativen Adressierung mit dem DRAW "B M"-Befehl:

siehe Kurzlisting DRAW 8 im Anhang

Gab es vorher mißmutige Spötter, die sich über die Bezeichnung, der DRAW-Befehl beinhalte eine vollständige Programmiersprache, aufgeregt haben? Ich hoffe, Sie können sich langsam beruhigen und finden diesen DRAW-Befehl in seiner vielseitigen Anwendungsmöglichkeit nun auch ein klein wenig phaszinierender.

Kommen wir zum nächsten DRAW-Unterbefehl: wir mischen einfach eine der 15 (bzw. 16) Farben in unsere Zeichnung mit ein. Dieser Vorgang wird mit dem Buchstaben "C" (für englisch COLOUR = Farbe) und einer nachfolgenden ein- bzw. zweiziffrigen Zahl ausgelöst. Damit alles wieder aus unserem vorherigen Beispiel in den Ursprungszustand versetzt wird, geben wir zu Beginn den Befehl DRAW "S4 A0" als eine Art Resetknopf ein:

siehe Kurzlisting DRAW 9 im Anhang

Nachdem wir unseren Grafikcursor wieder elegant "mittig" (128,96) positioniert haben, besagt "C15", daß ab jetzt in weißer Farbe gemalt wird und zwar 16 Punkte nach rechts; anschließend wird vor jeder Bewegung jeweils die Farbe geändert, so daß schließlich auf dem Bildschirm zwei schwarze ("C1") und zwei weiße ("C15") Balken unser Quadrat umschließen.

Den gleichen Effekt kann man erzielen, wenn man zwischenzeitlich immer von neuem den COLOR-Befehl benutzt, jedoch hier heißt es, VORSICHT! Andernfalls wird die Schriftfarbe auf dem Textbildschirm halt gleich miverändert (während der COLOR-Befehl nämlich auf Grafik und Text Einfluß ausübt, gilt dies bei Verwendung besonderer Grafik-Farbe-Unterkommandos halt nur auf die einzelne Grafik bezogen. Schauen wir uns trotzdem einmal an, wie unser obiges Programm unter Hinzuziehung des COLOR-Befehls ausschaut:

siehe Kurzlisting DRAW 10 im Anhang

Ganz schön lang ist das Programm nun geworden, obwohl es nichts anderes tut als unser Programm mit dem DRAW-C-Unterkommando zuvor. Wir müßten gar abschließend noch eine Zeile in der Form

COLOR 15

einfügen, um zur Grundfarbe zurückzukehren.

Noch eine Warnung! Hüten Sie sich davor, bei Ihrem MSX-Computer willkürlich die Farbe in zu kleinen Punktabständen zu verändern! Die MSX-Computer vertragen zwar, wie schon des öfteren angegeben, 256*192 Grafikpunkte, die Unterscheidung der Farbe spielt sich jedoch nur in einer Matrix von 32*192 ab, andernfalls überlagern sich die Farben wie Farbkleckse in dem folgenden Programm:

siehe Kurzlisting DRAW 11 im Anhang

Obgleich hier, wie in unseren vorherigen Beispiel, laut Programm zwei weiße und zwei schwarze Linien zu sehen sein sollten, ist dies leider Gottes aufgrund des Ineinanderlaufens der zu eng beieinander liegenden Farben nicht der Fall (Untergrundfarbe plus weiß plus schwarz d.h.: da ist mindestens eine Farbe zu viel).

Bis hierhin haben wir eigentlich alle Kommandos d.h. Buchstaben kennengelernt, die unserer DRAW-Programmiersprache innewohnen, als da sind:

R U L D	für die 90 Grad Richtungen
E F G H	für die 45 Grad Zwischenrichtungen
A	für den Drehfaktor um jeweils 90 Grad
M	für das Grafikcursorbewegen mit Zeichnung
B M	für das Grafikcursorbewegen ohne Zeichnung
C	für die Farbveränderung
S	für die Vergrößerung bzw. Verkleinerung
M 128,96	für die absolute Adressierung
M +50,-50	für die relative Adressierung

Zuletzt noch zu zwei Kommandos, die zwar keine DRAW-Unterbefehlserweiterungen sind, die uns aber das Programmieren der Grafiken mit Hilfe des DRAW-Befehls erheblich erleichtern.

1) Wir können Variablen in unsere DRAW-String einfließen lassen, indem wir folgendes Format anstelle einer Zahleingabe verwenden:

= Variablenname ;

Variablenname dürfte klar sein; vergessen Sie aber keinesfalls, das dem Variablennamen ihm vorhergehende Gleichheitszeichen und das ihm nachfolgende Semikolon zusätzlich einzugeben! Andernfalls wüßte Ihr MSX-Computer gar im Zweifelsfalle nicht darüber bescheid, was er zu tun hat, z.B. würde er bei Verwendung der Variablen A versuchen, die nachfolgende Zeichnung im 90 Grad Rhythmus zu drehen. Also geben Sie immer darauf acht, daß die Syntax bei der Eingabe stimmt, wie im folgenden Beispiel:

siehe Kurzlisting DRAW 12 im Anhang

Wau! Innerhalb von nicht einmal zwei Sekunden hat Ihnen Ihr MSX-Computer eine Vielzahl von Quadraten und immer größer werdenden Quadraten auf den Bildschirm gebracht. Während die innerste Zeichnung noch in Originalgröße aufgebaut wurde, nahm die Vergrößerungsrate stetig zu, bis schließlich mit "S=100" alles fünfundzwanzigmal so groß abgebildet wurde (d.h. 25*4 = anstelle von 4 Punkten wurden 100 Punkte gezeichnet - was sich zwischen kleinstem und größten Quadrat getan hat, können Sie am Bildschirm suksessive verfolgen).

Diese Variablenzuweisung mit "=" und ";" können wir in allen unseren DRAW-Unterkommandos anwenden, wie z.B. beim Zeichnen unserer Windrose bei Verwendung des Winkelbefehls mit "A":

siehe Programmlisting DRAW 13 im Anhang

Wie Sie bei dem letzten Programm sicherlich bemerkt haben, setzen wir am Ende (hier Zeile 60 des Programms) die veränderten Parameter auf ihren Normalstatus zurück. Dann kann es beim Neustart nicht zu Komplikationen kommen. Der Normalstatus heißt eigentlich vollständig:

A0 (Winkel = 0 Grad)
C15 (weiße Vordergrundfarbe)
S4 (keine Vergrößerung bzw. Verkleinerung)
B M 0,0 (Grafikcursor am Ausgangspunkt d.h. in der Ecke links oben)

2) Wir können bereits erstellte DRAW-Zeichenketten in neue DRAW-Zeichenketten als Unterzeichenketten einbauen.

Um dies zu demonstrieren, fangen wir nicht gleich an zu zeichnen, sondern erstellen uns ein Zeichentableau:

In Variable A$ speichern wir unsere Treppe vom Anfang ab ("R8 U8 R8"),

in Variable B$ unser Quadrat ("R8 U8 L8 D8") und schließlich

in Variable C$ unser Hexenhäuschen ("R8 U8 L8 E4 F4 G8 F8").

Die Syntax für die Verschachtelung der Stringvariablen wird mit dem Buchstaben "X" eingeleitet und wiederum mit einem Semikolon zur Kennzeichnung abgeschlossen:

X Variablenstring ;

Unsere Aneinanderkettung der drei Strings sieht dann letztendlich so aus:

siehe Kurzlisting DRAW 14 im Anhang

Zwar sieht das Ergebnis nicht gerade begeisternd aus, aber vielleicht zeigt es Ihnen in der verketteten Anwendung unserer zuvor gezeichneten Teilstücke zumindest, wie leistungsfähig und geradezu variabel man bei der Verkettung verschiedener DRAW-Teilstrings vorgehen kann.

Wie bereits zu Beginn dieses Kapitels angedeutet, hier noch einmal der Hinweis: Denken Sie daran, daß die DRAW-Strings genauso leicht zu behandeln sind wie normale Zeichenketten (A$="Hugo"). Also müssen Sie auch daran denken, daß ein String bei MSX-Computern "nur" 255 Zeichen lang sein darf! Außerdem müssen Sie eigentlich mit dem Befehl CLEAR vorher Speicherraum für die Strings reservieren (z.B. für vier Strings mit einer Länge von je 200 Zeichen müssen Sie vorher auf alle Fälle eingeben: CLEAR 800).

Abschließend noch ein paar kurze Hinweise:

1. Für die Zahlangabe zur Richtungsbestimmung sind Sie nicht dazu verpflichtet, nur Werte einzugeben, die sich im möglichen Bildrahmen darstellen lassen (256*192 Punkte). Vielmehr können Sie Werte zwischen +32767 und -32768 eingeben. Wofür dies jedoch gut sein soll, ist zumindest fraglich (eventuell könnte dazu ein Grundgedanke der MSX-Konstrukteure Anlaß gewesen sein, möglichst eine Vielzahl von Fehlermeldungen (und damit das unweigerliche Löschen des Grafikbildschirms) vermeiden zu helfen).

2. Sie können an vier Speicherstellen den aktuellen Grafikcursorstand ablesen bzw. einen neuen Grafikcursorstand eingeben (letzteres entspricht DRAW "B M X Y"):

&HFCB3	X-Grafikcursor, absolut
&HFCB5	Y-Grafikcursor, absolut
&HFCB7	X-Grafikcursor, relativ
&HFCB9	Y-Grafikcursor, relativ

3. An Speicherstelle

&HFCBC

ist der aktuelle DRAW-Skalen=Vergrößerungs-/Verkleinerungsfaktor abgespeichert bzw. kann von Ihnen auch dort verändert werden.

4. An Speicherstelle

&FCBD

ist der aktuelle DRAW-Angel=Winkelfaktor abgespeichert bzw. kann von Ihnen auch dort verändert werden.

5. Schließlich ist an Speicherstelle

&HF3E9

die aktuelle Vordergrundfarbe abgespeichert bzw. kann ebenfalls von Ihnen auch dort verändert werden.

DRAW 1

```
              -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 DRAW "B M 128,96 U8 R8 U8 R8 D16 L16"
40 GOTO 40
```

DRAW 2

```
              -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 DRAW "B M 128,96 R8 U8 L8 E4 F4 G8 U8
 F8"
40 GOTO 40
```

DRAW 3

```
              -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
```

```
30 DRAW "B M 128,96 R8 U8 L8 D8"
40 DRAW "B M 128,96 S8Ø R8 U8 L8 D8"
50 GOTO 5Ø
```

DRAW 4

-*-*-*-*-*-

```
1Ø COLOR 1,15
2Ø SCREEN 2
3Ø DRAW "B M 128,96 R8 U8 R8"
4Ø GOTO 4Ø
```

DRAW 5

-*-*-*-*-*-

```
1Ø COLOR 1,15
2Ø SCREEN 2
3Ø DRAW "B M 128,96 U16 F4 H4 G4"
4Ø N=N+1
5Ø ON N GOSUB 7Ø,9Ø,11Ø
6Ø GOTO 6Ø
7Ø DRAW "A1"
8Ø GOTO 3Ø
9Ø DRAW "A2"
1ØØ GOTO 3Ø
11Ø DRAW "A3"
12Ø GOTO 3Ø
```

DRAW 6

-*-*-*-*-*-

```
1Ø COLOR 1,15
2Ø SCREEN 2
3Ø DRAW "B M 128,96 N AØ U8 N A1 U8 N A2
 U8 N A3 U8"
4Ø GOTO 4Ø
```

DRAW 7

-*-*-*-*-*-

```
1Ø COLOR 1,15
2Ø SCREEN 2
3Ø DRAW "B M 128,96"
4Ø FOR N=1 TO 4
5Ø DRAW "B M +Ø,+8 R4 U4 L4 D4"
6Ø NEXT N
7Ø GOTO 7Ø
```

DRAW 8

-*-*-*-*-*-

```
1Ø COLOR 1,15
2Ø SCREEN 2
3Ø DRAW "S4 AØ B M 5Ø,13Ø"
4Ø FOR N=1 TO 8
5Ø DRAW "B M +18,+Ø R8 U4 L3 U12 R3 U2 L
4 E3 H3 G3 F3 L4 D2 R3 D12 L3 D4
6Ø NEXT N
7Ø DRAW "S4 AØ B M 5Ø,66"
8Ø FOR N=1 TO 8
9Ø DRAW "B M +16,+2 R8 U4 L3 U12 R3 U2 L
4 E3 H3 G3 F3 L4 D2 R3 D12 L3 D4
1ØØ PAINT STEP(+2,-2)
11Ø NEXT N
12Ø GOTO 12Ø
```

DRAW 9

-*-*-*-*-*-

```
10 COLOR 1,4
20 SCREEN 2
30 DRAW "S4 AØ"
40 DRAW "B M 128,96 C15 R16 C1 U16 C15 L
16 C1 D16"
50 GOTO 50
```

DRAW 10

-*-*-*-*-*-

```
10 COLOR 1,4
20 SCREEN 2
30 DRAW "S4 AØ"
40 DRAW "B M 128,96"
50 COLOR 15
60 DRAW "R 16"
70 COLOR 1
80 DRAW "U 16"
90 COLOR 15
100 DRAW "L 16"
110 COLOR 1
120 DRAW "D 16"
130 GOTO 130
```

DRAW 11

-*-*-*-*-*-

```
10 COLOR 1,4
20 SCREEN 2
30 DRAW "B M 128,96 C15 R4 C1 U4 C15 L4
C1 D4"
40 GOTO 40
```

DRAW 12

-*-*-*-*-*-

```
10 COLOR 1,4
20 SCREEN 2
30 FOR N=4 TO 100 STEP 4
40 DRAW "B M 128,96 S=N;R4 U4 L4 D4"
50 NEXT N
60 DRAW "S4"
70 GOTO 70
```

DRAW 13

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=Ø TO 3
40 DRAW "B M 128,96 A=N;U8"
50 NEXT N
60 DRAW "AØ"
70 GOTO 70
```

DRAW 14

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 2
30 A$="S4 R8 U8 R8"
40 B$="R8 U8 L8 D8"
50 C$="S8 R8 U8 L8 E4 F4 G8 U8 F8"
60 DRAW "B M 128,96 X A$;X B$;X C$;"
70 GOTO 70
```

Befehle Grafik PSET

Zeichnen eines farbigen Punktes auf die Grafikbildschirme

siehe hierzu auch die Programme im Anhang: POS 6
PRESET 1

Der Befehl PSET gehört zu den Grafikbefehlen, die nur auf SCREEN 2 und SCREEN 3 zu einer sinnvollen Funktion führen. Würden wir PSET auf den Textbildschirmen SCREEN 0 und 1 anzuwenden versuchen, würde als Ergebnis die Errormeldung lauten: Illegal function call.

Das Format des Kommandos PSET lautet:

```
PSET(X,Y),Z
```

wobei X und Y die Koordinaten zum Punktsetzen sind, während mit Z die Punktfarbe angegeben wird (s.u.). Wir haben bei PSET wie auch bei allen anderen Grafikkommandos die Möglichkeit, den gewählten Punkt (Angabe durch die X- und Y-Koordinate) absolut oder relativ zu adressieren. Absolut heißt, daß die Koordinatenangaben sich auf die verfügbare Punktmatrix von 256*192 Grafikpunkten direkt bezieht (z.B. würde die Angabe 128,96 bedeuten, daß der Punkt in die Bildschirmmitte gesetzt wird).

Eine andere Möglichkeit der Koordinatenansprache beim PSET-Befehl ist die relative Adressierung. Zu diesem Zweck setzen wir vor die X-Y-Angabe das Wörtchen STEP, was dann folgendes bedeutet: Gehe vom zuletzt angesprochenen Grafikpunkt gemäß der Angabe bei X,Y entsprechend viele Punkte nach rechts-links (X-Angabe) bzw. nach oben-unten (Y-Angabe). An einem Beispiel gezeigt bedeutet das folgendes:

PSET(128,96)
setze einen Punkt an Koordinate 128,96

PSET STEP(10,10)
setze einen Punkt von der letzten Koordinate 128,96
10 Punkte nach rechts (138) und
10 Punkte nach unten (106)

Mit der Z-Angabe im PSET-Befehl haben wir die Möglichkeit, eine Punktfarbe auszuwählen. Wird Parameter Z hingegen nicht angegeben, übernimmt Ihr MSX-Computer die aktuelle Vordergrundfarbe zum Zeichnen des Punktes.

Es ist auch möglich, den PSET-Befehl so zu gebrauchen, daß er wie das BASIC-Kommando PRESET arbeitet. Dazu müssen wir uns über die Unterschiede von PSET und PRESET klarwerden: Während bei PSET ohne Farbangabe die aktuelle Vordergrundfarbe verwendet wird, ist es entsprechend beim PRESET-Befehl die aktuelle Hintergrundfarbe. Da der Großteil des Grafikbildschirms häufig von der Hintergrundfarbe bestimmt wird, kommt somit die Verwendung des PRESET-Befehls einem punktweisen Bildlöschen gleich. Um diese Funktion auch mit PSET durchführen zu können, müssen wir als Punktfarbe die Hintergrundfarbe eingeben. Diese entsprechende Farbnummer finden wir zwar nicht durch einen passenden MSX-Grafikbefehl, wir müssen vielmehr Speicherstelle &HF3EA befragen, die uns die entsprechende Auskunft vermittelt. Also gilt programmtechnisch folgendes:

```
PSET(X,Y),PEEK(&HF3EA)=PRESET(X,Y)
```

Um den PSET-Befehl so zu verwenden, als wäre es PRESET, können wir auch ohne Speicherabfrage vorgehen, indem wir die zuletzt geänderte Hintergrundfarbe in einer Variablen zwischenspeichern. Geschieht dies immer, können wir die entsprechende Variable an den PSET-Befehl zur Hintergrund-Farbgebung anhängen.

Sie sind nicht dazu verpflichtet, Punkte nur auf der darstellbaren Bildschirmfläche (256*192 Punkte) zu zeichnen. Vielmehr können Sie sowohl für X als auch für Y Werte zwischen -32768 und +32767 einsetzen, wobei man sich allerdings fragen sollte: Wofür? Werden noch größere bzw. noch kleinere Werte für X bzw. Y eingegeben, erfolgt die Fehlermeldung: 'Overflow'.

Wie auch bei allen anderen Grafikbefehlen bewirkt das Punktsetzen mit PSET eine Aktualisieung des Grafikcursors. Sie können diese aktuelle Koordinate jederzeit abrufen (aber auch durch POKE wieder ändern), wenn Sie sich die Speicherstellen &HFCB3, &HFCB5, &HFCB7 und &HFCB9 einnmal etwas genauer ansehen:

Speicherstelle &HFCB3: absolute X-Koordinate
Speicherstelle &HFCB5: absolute Y-Koordinate
Speicherstelle &HFCB7: relative X-Koordinate
Speicherstelle &HFCB9: relative Y-Koordinate

Zum einen wird der aktuelle Grafikcursor somit in den Speicherstellen &HFCB3 und &HFCB5 festgehalten, während die Speicherstellen &HFCB7 und &HFCB9 darüber Auskunft geben, wie weit sich im Augenblick der Grafikcursor vom eigentlichen Cursor entfernt hat (Beispiel: Kreiszeichnung durch Mittelpunktangabe, STEP verwendet im PSET-Befehl).

siehe Kurzlisting POS 6 im Anhang

Noch zwei Hinweise:

1) Wollen Sie nur den Grafikcursor aktualisieren (ähnlich dem LOCATE-Befehl auf den Textbildschirmen) können Sie dies über den Befehl

PRESET(X,Y) oder
PSET(X,Y),Hintergrundfarbe

am besten und sichsersten tun. Die Befehlseingabe mit PSET allein (ohne Farbangabe) wäre hingegen nicht sehr sinnvoll, da wir als Folge davon augenblicklich an der Zielstelle über einen nicht gewollten Bildschirmpunkt verfügen würden. Hingegen wird mit PRESET der Anfangspunkt unsichtbar auf die Hintergrundfarbe eingestellt.

siehe Kurzlisting PRESET 1 im Anhang

Ihr MSX-Computer verfügt zwar über eine hochauflösende Grafik von 256*192 Punkten, jedoch läßt die Farbgrafik in diesem Rahmen stark zu wünschen übrig: Sie haben nur 32*192 Farbbildpunkte (horizontale Blöcke) zur Verfügung. Dadurch bedingt müssen Sie bei einer Grafikdarstellung mit Vorsicht darauf achtgeben, daß nicht zwei verschiedene Vordergrundfarben in einem Feld von 8*1 Punkten zusammentreffen - sonst kommt es zur Farbüberlappung (die Farben laufen dann anscheinend unkontrolliert ineinander).

Befehle Grafik PRESET

Löschen eines Bildschirmpunktes auf den Grafikbildschirmen

siehe hierzu auch die Programme im Anhang: PRESET 1
PRESET 2

Wollen wir auf den Grafikbildschirmen SCREEN 2 bzw. SCREEN 3 einen Punkt auf den Bildschirm zeichnen, bedienen wir uns dabei des Befehls PSET. Durch die Angabe der X-Y-Kordinaten sowie der Farbnummer (0 bis 15) können wir gezielt vorgehen.

Manchmal ist es aber auch sinnvoll, einen oder gar mehrere Punkte auf dem Grafikbildschirm wieder zu löschen. Wir müssen uns hierbei darüber klar sein, was unter 'löschen' verstanden werden soll: Entweder setzen wir die zu löschenden Bildschirmpunkte = der Hintergrundfarbe oder übermalen sie mit der einer neuen Vordergrundfarbe.

Bei der Anwendung des Punktlöschbefehls PRESET sind beide Möglicheiten vorgesehen:

PRESET(X,Y) = Löschen des Bildschirmpunkts bei X,Y
PRESET(X,Y),Z = Übermalen des Bildschirmpunkts bei X,Y mit der Farbe Z

Schauen wir uns diese 'Radiergummifunktion' des PRESET-Befehls einmal im folgendem Programm an:

siehe Kurzlisting PRESET 1 im Anhang

Erst wurde ein schwarzes (Farbe 1) Viereck auf weißen (Farbe 15) Untergrund gemalt (Zeilen 30 bis 60). Anschließend wurde das schwarze Viereck mit der Untergrundfarbe (= 15 = weiß) übermalt (Zeilen 70 bis 110).

Die gleiche Funktion wie PRESET läßt sich mit dem Kommando PSET und einer Farbangabe (die der Hintergrundfarbe entspricht) durchführen. Sie können also statt des PRESET-Befehls in Zeile 90 auch schreiben:

```
90 PSET(M,N),15
```

Ebenso läßt sich der PSET-Befehl in Zeile 50 durch ein entsprechendes PRESET mit Farbangabe ersetzen. Die Farbangabe muß in diesem Fall unserer gewählten Vordergrundfarbe entsprechen:

```
50 PRESET(M,N),1
```

Genaugenommen erfüllt also der PSET-Befehl die gleichen Funktionen wie der PRESET-Befehl; entsprechend umgekehrt gilt: Der PRESET-Befehl erzeugt die gleichen Funktionen wie der PSET-Befehl. Einziger Vorteil der 2 unterschiedlichen Befehle ist es, daß sowohl bei PSET als auch bei PRESET auf die Farbangabe (bei PSET die Vordergrundfarbe, bei PRESET die Hintergrundfarbe) verzichtet werden darf, während bei der umgekehrten Anweisung diese Farbangabe dringendst erforderlich ist.

Um dennoch bei einem Befehl bleiben zu können (entwder PSET oder PRESET) können wir uns an folgenden 2 Speicherstellen orientieren:

bei &HF3E9 ist die Vordergrundfarbe gespeichert
bei &HF3EA ist die Hintergrundfarbe gespeichert

Sie können mit Hilfe dieser Speicherstellen also PSET und PRESET auf folgende Art und Weise ersetzen:

```
PSET(X,Y) = PRESET(X,Y),PEEK(&HF3E9)
PRESET(X,Y) = PSET(X,Y),PEEK(&HF3EA)
```

Die Ansteuerung von Punkten kann auf den Grafikbildschirmen (SCREEN 2 und SCREEN 3) mit absoluter und relativer Adressierung vorgenommen werden. Während bei absoluter Eingabe die Koordinaten direkt angesprochen werden (z.B. 128,96 ist der Bildschirmmittelpunkt), gehen wir bei der relativen Adressierung vom

derzeitigen Grafikcursor aus und akzeptieren ihn als neuen Nullpunkt. Das Vorgehen vom Nullpunkt wird mit positiven bzw. negativen X- und Y-Werten angegeben. Um die relative von der absoluten Adressierung zu unterscheiden, müssen wir bei relativer Adresseneingabe das Wörtchen STEP mit eingeben:

```
PRESET STEP(10,-10)
```

Das bedeutet: Der nächste Punkt wird vom aktuellen Cursor gerechnet, um 10 Punkte nach rechts (10) und 10 Punkte nach oben (-10) gesetzt und gelöscht. Die Farbe mit der gelöscht wird ist hierbei die aktuelle Hintergrundfarbe bzw. bei PSET wäre es die Vordergrundfarbe.

Der Eingabebereich des PRESET-Befehls beschränkt sich nicht nur auf die sichbaren Koordinaten, nein wir können Werte zwischen -32768 und +32767 eingeben. Werte größer als diese Zahlen rufen einen 'Overflow error' hervor

Erlaubte Farben sind 0 bis 15; die Farbe 0 bedeutet durchsichtig und bewirkt somit keine offensichtliche Änderung auf dem Bildschirm. Werte höher als 15 und kleiner als 0 erzeugen einen 'Illegal function call'-Error.

Jede PRESET-Eingabe aktualisiert den Grafikcursor, so daß

```
PRESET(128,96)
CIRCLE STEP(0,0),30
```

bedeutet, daß der Kreis bei der Koordinaten 128,96 seinen Mittelpunkt hat.

Der PRESET-Befehl ermöglicht es uns aufgrund seiner unsichtbaren Grafikcursoränderung (ähnlich wie DRAW"B M X,Y") bei Textausgabe auf dem Grafikbildschirm die Anfangskoordinaten zu bestimmen:

siehe Kurzlisting PRESET 2 im Anhang

In diesem Punkt kommt dem PRESET-Befehl eine ähnliche Bedeutung wie dem Kommando LOCATE auf den Textbildschirmen zu.

PRESET 1

```
              -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 FOR N=100 TO 120
40 FOR M=80 TO 100
50 PSET(M,N)
60 NEXT M,N
70 FOR N=100 TO 120
80 FOR M=80 TO 100
90 PRESET(M,N)
100 NEXT M
110 NEXT N
120 GOTO 120
```

PRESET 2

```
              -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 OPEN "GRP:" FOR OUTPUT AS #1
40 PRESET(128,96)
50 PRINT #1,"MSX";
60 CLOSE
70 GOTO 70
```

Befehle Grafik POINT
Farbabfrage der einzelnen Grafikpunkte

siehe hierzu auch die Programme im Anhang: POINT 1
POINT 2
POINT 3
Programm 15

Wenn wir auf dem hochauflösenden Grafikbildschirm SCREEN 2 oder auch auf dem niedrigauflösenden Grafikbildschirm SCREEN 3 arbeiten, befinden wir uns in einem Koordinatensystem mit 256*192 bzw. 64*48 Punkten.

Während bei SCREEN 3 jeder Punkt (oder sagen wir lieber Block) eine der zur Verfügung stehenden 16 Farben (inklusive der "Farbe" durchsichtig) annehmen kann, sieht dies bei SCREEN 2 ein bißchen diffiziler aus:

SCREEN 2 bietet zwar die hochauflösende Grafik von 256*192 Punkten, was bedeutet, daß knapp 50000 verschiedene Punkte auf dem Bildschirm zur Grafikdarstellung angesteuert werden können, aber mit der Farbauflösung hapert es doch noch ein klein wenig: Es können zwar alle 16 Farben in der Vertikalen Punkt unter Punkt mit verschiedenen Farben belegt werden, ohne daß es zu Überschneidungen kommt; in der Horizontalen ist dies allerdings nur in einer Folge von 8 Punkten möglich (Punkt 0 bis 7 = 2 verschiedene Farben, Punkt 8 bis 15 = 2 verschiedene Farben usw.). Versuchen wir, eine dritte Farbe einzumischen, so wird sie zur dominierenden Vorder- bzw. Hintergrundfarbe, was sich auch auf bereits bestehende andersfarbige Punkte auswirkt.

Mit dem POINT-Befehl haben wir die Möglichkeit, den Bildschirm Punkt für Punkt auf Farbe zu untersuchen. Dabei müssen wir nur die entsprechenden X-Y-Koordinaten eingeben und zur Entschlüsselung der ausgegeben Zahl in einer Farbtabelle nachsehen. Dazu ein Beispiel:

siehe Kurzlisting POINT 1 im Anhang

Da wir in Zeile 30 ein Rechteck mit schwarzer Farbe (=1) ausgefüllt haben, ist auch jeder Punkt innerhalb dieses Rechtecks samt der Rechteckumrandungsfarbe schwarz.

Im folgenden Beispiel lesen wir uns den Inhalt des Grafikbildschirms in die dimensionierte Variable A ein und zeigen das Grafikabbild anschließend in Form von Farbzahlwerten auf dem Textbildschirm:

siehe Kurzlisting POINT 2 im Anhang

In Zeile 50 wird ein kleines Rechteck in der Größe 8*8=64 Felder auf dem Bildschirm aufgetragen. In den darauffolgenden Zeilen wird nun die Grafikbildschirminformation Punkt für Punkt in die dimensionierte Variable A eingelesen, deren Werte anschließend der Reihe nach auf dem Textbildschirm wieder - nun aber in Form von Farbzahlwerten - deutlich werden lassen, daß es sich bei der eingelesenen Bildschirmfläche um unser Rechteck handelt. Während das Rechteck mit Farbe 2 gemalt und ausgefüllt wurde, ist die Umrandungsfarbe Farbe Nummer 3 (wie im COLOR-Befehl in Zeile 10 definiert).

Mit Hilfe einiger kleiner Änderungen in obigem Programm können Sie sich auch die Form von Buchstaben und Zeichen punktweise vom Grafikbildschirm ablesen und vergrößert auf dem Textbildschirm zur Darstellung bringen. Schauen wir uns zu diesem Zweck einmal an, wie der Buchstabe "A" Punkt für Punkt aussieht:

siehe Kurzlisting POINT 3 im Anhang

Da der Buchstabe 'A' in der Vordergrundfarbe=1=schwarz gemalt worden ist, können Sie an den ausgegebenen Einsen die eingelesene Form des 'A' wiedererkennen.

Geben Sie ruhig andere Buchstaben oder Zeichen in Zeile 70 ein. Mit ein wenig Phantasie können Sie es sich somit vielleicht sogar sparen, den in diesem Buch abgedruckten Charactergenerator (siehe Programm 14) zur Zeichenerstellung heranzuziehen. Sie können vielmehr mit Hilfe des gerade eingetippten Programms etwas ähnliches erzielen.

Wozu läßt sich der POINT-Befehl anwenden, außer zu unseren kleinen Spielchen, die wir bisher vorgenommen haben?

1) Für die Ausgabe des hochauflösenden Bildschirms auf einen angeschlossenen Nadel- oder Matrixdrucker müssen wir den Bildschirm Punkt für Punkt abtragen - eine ideale Aufgabe, die sich mit Leichtigkeit mit POINT ausführen läßt.

siehe Programm 15 im Anhang

Bedenken Sie aber beim Schreiben einer Hardcopyroutine, daß nur bestimmte Farben auf den Drucker ausgegeben werden sollen, z.B. durch den Befehl:

```
IF POINT(X,Y)=3 THEN LPRINT ...
```

gibt nur dann Farbe auf den Drucker aus, wenn die Bildschirmfarbe = 3 ist, denn die meisten Matrixdrucker haben ja nur zwei Möglichkeiten: entweder 'schwarz' drucken oder 'weiß' belassen - wir müssen also ein Farbbild in ein Schwarzweißbild übertragen.

2) Bei einem Spiel ist es manchmal nötig, Informationen vom Bildschirm im Programm abzulesen z.B. ein Männchen wandelt auf einer roten Mauer und muß über Löcher springen. Da diese Löcher sicherlich nicht 'rot' sind, können wir die Schritte des Männchens mit Hilfe des POINT-Befehls auf Standfestigkeit ja/nein (unter den Füßen des Männchens ist die Farbe rot) untersuchen.

Noch ein paar Hinweise zum sinnvollen Gebrauch des POINT-Befehls:

1. Sie können den POINT-Befehl sinnvoll bei Anwahl von SCREEN 2 und SCREEN 3 zur Abfrage der Bildschirmpunktfarbe einsetzen. Auch bei SCREEN 0 und SCREEN 1 werden zwar Werte ausgegeben; diese Werte sind allerdings z.B. zum Lesen der vorher auf SCREEN 2 erstellten Grafik nicht sinnvoll nutzbar. Vielmehr handelt es sich hier um die bitweise Abfrage eines Speicherbereichs, der bei der Bildschirmdarstellung von SCREEN 0 und SCREEN 1 keine Bedeutung hat.

2. Sie können (sinnvoll) mit POINT 'nur' das Grafikbildschirmfeld von 256*192 Punkten nach der Farbbelegung befragen. Jedoch ist es auch möglich, Werte zwischen -32768 und +32767 (größere bzw. kleinere Werte erzeugen einen 'Overflow'-Error) für die X- bzw. die Y-Koordinate des POINT-Befehls einzugeben. Die Ausgabe nach Abfrage des POINT-Befehls lautet in diesen Fällen: -1.

3. Sie können den POINT-Befehl auch durch direkte Nachfrage im VRAM durch das BASIC-Kommando VPEEK ersetzen. Jedoch gehört hierzu ein genaues Wissen über den Aufbau des VRAM (siehe unter SCREEN, BASE und VDP) und ein recht komplexes Speicherentschlüsselungsprogramm, zumindest dann, wenn es SCREEN 2 betrifft: Hier setzt sich die Farbinformation aus Daten des Colour Table und des Pattern Name Table zusammen.

4. Abschließend noch eine weitere Anwendung des POINT-Kommandos: Um sich selbst eine PAINT-Rand-Prüfroutine zu erstellen, können Sie mit POINT die entsprechende Umrandungsfarbe abfragen und dann mit PAINT ohne Befürchtung die Fläche auszumalen beginnen.

POINT 1

```
                    -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 LINE(128,96)-(148,76),1,BF
40 A=POINT(138,85)
50 FOR N=1 TO 1000
60 NEXT N
70 SCREEN 0
80 PRINT A
```

POINT 2

```
                    -*-*-*-*-*-
10 COLOR 1,3
20 SCREEN 2
30 WIDTH 37
40 DIM A(12,12)
50 LINE(128,96)-(135,103),2,BF
60 FOR N=127 TO 136
70 FOR M=95 TO 106
80 A(N-126,M-94)=POINT(N,M)
90 NEXT M
100 NEXT N
110 SCREEN 0
120 FOR N=1 TO 10
130 FOR M=1 TO 10
140 PRINT A(M,N);
150 NEXT M
151 PRINT
160 NEXT N
```

POINT 3

```
                    -*-*-*-*-*-
10 COLOR 1,3
20 SCREEN 2
30 WIDTH 37
40 DIM A(10,10)
50 OPEN "GRP:" FOR OUTPUT AS #1
60 DRAW "BM 127,97"
70 PRINT #1,"A"
80 FOR N=127 TO 136
90 FOR M=97 TO 106
100 A(N-126,M-96)=POINT(N,M)
110 NEXT M
120 NEXT N
130 SCREEN 0
140 FOR N=1 TO 10
150 FOR M=1 TO 10
160 PRINT A(M,N);
170 NEXT M
180 PRINT
190 NEXT N
200 CLOSE
```

Zeichnen von Linien und Boxen in hochauflösender Grafik

siehe hierzu auch die Programme im Anhang: LINE 1
LINE 2
LINE 3
LINE 4
LINE 5
LINE 6
LINE 7
LINE 8
LINE 9
LINE 10
LINE 11
LINE 12
LINE 13
DRAW 6

Der MSX-Grafikbefehl LINE hat eine dreifache Bedeutung:

1) Wir können Linien zeichnen
2) Wir können Rechtecke malen
3) Wir können Rechtecke malen und sie mit Farbe ausfüllen

zu 1: Sie haben einerseits die Möglichkeit, sowohl den Anfangscursor als auch den Endcursor anzugeben, andererseits können Sie auch den Anfangscursor von seiner letzten Position her übernehmen und lediglich bestimmen, wo die Linie hingehen soll.

Bei letztgenannter Möglichkeit müssen Sie jedoch darauf achtgeben, daß sich Ihr MSX-Computer im Anfangsstadium (nach dem Einschalten) in der Ecke links oben bei Position 0,0 befindet. Doch schauen wir uns beide Möglichkeiten einmal in folgendem Programm an:

siehe Kurzlisting LINE 1 im Anhang

In Zeile 30 erfolgt der Befehl, die zu zeichnende Linie in der Bildschirmmitte bei Cursorposition 128,96 zu beginnen und sie bis zur Koordinate 150,150 durchzuziehen. Da anschließend in Zeile 40 der Linienbeginn nicht angegeben ist, übernimmt Ihr MSX-Computer ganz einfach die zuletzt angesteuerte Koordinate als neuen Anfangscursor (150,150) und zeichnet von dort die Linie weiter bis Cursorposition 160,120. Würden wir Zeile 30 wie folgt abändern:

```
30 LINE-(150,150)
```

und das bestehende Programm noch einmal neu starten, zeichnet Ihr Computer von der beim vorigen Programmablauf zuletzt angesprochenen Cursorposition (160,120) und zieht die Linie bis 150,150 durch (siehe Zeile 30).

Um allerdings wieder in den Anfangszustand zu gelangen, müssen wir zu Beginn noch eine Zeile einfügen, damit der Grafikcursor wieder bei 0,0 (wie nach dem Neueinschalten) mit dem Zeichnen beginnen kann:

```
25 DRAW "B M 0,0"
```

Nun startet der Grafikcursor in Koordinate 0,0 und zieht eine Linie bis zum Punkt (wie in Zeile 30 angegeben) 150,150.

Sie müssen sich also immer wieder in Erinnerung rufen, bevor mit der Zeichnung begonnen wird, wo sich der Grafikcursor zur Zeit befindet, damit es beim einen Programmablauf nicht zu unerwarteten Überlagerungen kommt (nur weil der Cursor an der falschen Bildschirmposition zu zeichnen beginnt!).

Neben der absoluten Adressierung (im Rahmen von 256 * 192 möglichen Bildschirmpunkten wird genau angegeben, wo mit der Zeichnung begonnen und an welcher Koordinate mit der Zeichnung aufgehört werden soll) kann beim LINE-Befehl auch die relative Adressierung mit STEP angewendet werden:

siehe Kurzlisting LINE 2 im Anhang

Obwohl diese Zeichnung mehr willkürich zusammengestellt wurde, hier die Erklärung - Zeile für Zeile:

Zeile 40: absolut gesetzt wird vom Bildschirmmittelpunkt (128,96) bis zur Koordinate 150,150 eine Linie gezeichnet

Zeile 50: wir befinden uns an der Bildschirmposition 150,150. Der vorgesetzte STEP-Befehl besagt nun, daß der Grafikcursor vom zuletzt angewählten Bildschirmpunkt (150,150) um 10 Punkte nach rechts und 10 Punkte nach unten gehen soll, bevor mit dem Zeichnen begonnen wird. Von dort soll die Linie wiederum absolut (weil ohne STEP) zur Bildschirmkoordinaten 180,180 führen.

Zeile 60: da hier sowohl der Anfangspunkt der neuen Linie als auch ihr Endpunkt vorneweg die Angabe STEP stehen haben, wird hier nur relativ vorgegangen und zwar liegt der Linienanfang (-20,-20) um 20 Punkte nach links und 20 Punkte nach oben vom zuletzt eingegebenen Cursorounkt versetzt. Von hier aus führt die Linie weiter um -40,-40 d.h. um 40 Punkte nach links und 40 Punkte nach oben.

Da die MSX-Computer für einen Mathematiker ungewohnt mit der Zeichnung links oben (Cursorposition 0,0) beginnen, muß man sich die Plus- und Minusrichtungen bei relativer Adressierung der Übersichtlichkeit halber einmal genauer übersetzen:

```
LINE STEP(x1,y1)-(x2,y2)
```

Wenn x1 positiv (z.B. 3,3) -° weiter nach rechts
Wenn x1 negativ (z.B.-3,3) -° weiter nach links
Wenn y1 positiv (z.B. 3,3) -° weiter nach unten
Wenn y1 negativ (z.B.3,-3) -° weiter nach oben

Würde das MSX-Betriebssystem mathematisch-geometrisch richtig vorgehen (Orientierung am Koordinatensystem) müßte eigentlich links unten der Bildschirmpunkt 0,0 liegen. Die äußerste Ecke (255,191) hingegen dürfte nicht, wie bei den MSX-Computern, rechts unten liegen, eigentlich wäre die Ecke rechts oben viel passender. Aber da dies nun einmal nicht so ist (obgleich mathematisch logischer), müssen wir uns an oben aufgeschriebener Tabelle orientiern.

Bereits mit dem LINE-Befehl, wie wir ihn bis hierher kennengelernt haben, läßt sich schon allerhand anstellen: Sie können z.B. Zeichnungen von Figuren entwerfen ...

Vorteil gegenüber dem DRAW-Befehl: Wir brauchen nicht nur in acht vorgegebene Richtungen Linien zu zeichnen, wir können jeden beliebigen Zeichenwinkel ansprechen. Probieren wir dies einfach einmal aus:

siehe Kurzlisting DRAW 6 im Anhang

Falls Sie sich mit dem Kapitel über den DRAW-Befehl bereits beschäftigt haben, wird Ihnen die in diesem Programm erstellte Windrose noch ein Begriff sein.

Mit Hilfe des LINE-Befehls aber können wir, wie bereits gesagt, jeden Winkel anspringen. Dadurch sieht man im folgenden Programm schließlich eine ausgemalte Fläche:

siehe Kurzlisting LINE 3 im Anhang

Jede nur mögliche Richtung wird suksessive (Zeile 30) angesprungen, so daß schließlich ein ausgemaltes Dreieck entsteht.

Um zu zeigen, daß wir auch Strahlen in größerem Abstand mit dem LINE-Befehl (ähnlich dem DRAW-45-Grad-Kommando) zeichen können, zeigt schließlich noch dieses Programm:

siehe Kurzlisting LINE 4 im Anhang

Nach und nach wird ein Strahlenbild aufgebaut, bis sich die Strahlen (im STEP 1-Abstand wie in unserem vorigen Programm) zu einer ausgemalten Fläche vereinigen.

zu 2: Wir können mit Hilfe des LINE-Befehls nicht nur eine Linie von x1,y1 bis x2,y2 ziehen, sondern auch Anfangs- und Endpunkt dieser Angaben als Umschreibung eines Rechtecks verwenden.

Schauen wir uns erst einmal in einem kleinen Programm an, wie es möglich ist, mit Hilfe des normalen LINE-Befehls ein Rechteck zu zeichnen:

siehe Kurzlisting LINE 5 im Anhang

Dies wäre das entsprechende Programm dazu mit absoluter Adressierung (bei unserem derzeitigen Wissensstand, wie mit dem LINE-Befehl ein Rechteck gezeichnet werden kann). Wenn Sie die eingetragenen Werte in Zeile 30 bis 60 einmal etwas genauer betrachten, so können Sie feststellen, daß wir zur Ansteuerung der vier verschiedenen Punkte eigentlich nur 2 x- und 2 y-Werte benötigten und zwar:

x1=128 x2=148
y1=96 y2=76

Schauen wir uns einmal an, welche Linie auf dem Bildschirm aufgetragen wird, wenn wir alle diese Koordinaten in einem LINE-Befehl verwenden:

siehe Kurzlisting LINE 6 im Anhang

Es ist die Diagonale, die unser vorhin gezeichnetes Rechteck von links unten nach rechts oben quer durchschneidet.

Aus diesen zwei Koordinaten (128,96 und 148,76) kann sich nun unser MSX-Computer ohne Probleme die zwei anderen Koordinaten oder Ecken unseres zukünftigen Rechtecks ausrechnen:

Es existieren bereits die Koordinaten:
128,96 148,76

Es werden aus diesen Koordinaten errechnet:
128,76 148,96

Hierbei führt der MSX-Computer selbstverständlich auch noch die automatische Zuweisung in der Reihenfolge so durch, daß schließlich auf dem Bildschirm ein Rechteck entsteht:

1)	128,96		128,96
2)	128,76	oder	148,96
3)	148,76		148,76
4)	148,96		128,76

Denn bei einer falschen Reihenfolge der gewählten vier Koordinaten würde das Rechteck teilweise aus Diagonalen, teilweise aber auch aus passenden Grundlinien bestehen.

Im englischen Sprachraum bezeichnet man ein Rechteck auch als "Box". Aus diesem Grund haben sich die Entwickler des MSX-BASIC einfach (wie auch beim DRAW- und beim PLAY-Befehl) den Anfangsbuchstaben von Box zum Erweitern des LINE-Befehls genommen; so heißt das Kommando zum Zeichnen eines Rechtecks oder einer Box:

```
LINE(x1,y1)-(x2,y2),,B
```

oder, wenn der Anfangspunkt bereits bekannt ist:

```
LINE-(x2,y2),,B
```

Bitte geben Sie acht, daß vor den Buchstaben "B" zwei Kommata gesetzt werden. Warum dies so ist, darüber später mehr.

Schauen wir uns im folgenden Beispielprogramm einmal an, wie schnell unser MSX-Computer eine Vielzahl von Vierecken oder Rechtecken auf den Bildschirm zeichnen kann:

siehe Kurzlisting LINE 7 im Anhang

Beachtlich schnell, wie hier gleich 1000 Rechtecke auf den Bildschirm gezeichnet werden!

Ohne näher darauf eingehen zu wollen, darf hier natürlich der Hinweis nicht fehlen, daß wir selbstverständlich auch mit Hilfe des STEP-Befehls (wie bereits unter 'zu 1': ausführlich erläutert) relativ (der augenblickliche Grafikcursor wird als 0,0 angesehen) Rechtecke zeichnen können. Das folgende Programmbeispiel soll dies verdeutlichen:

siehe Kurzlisting LINE 8 im Anhang

Hier wird nun zweierlei deutlich:

1) Die Angabe der Diagonalen (links unten nach rechts oben) reicht aus, um ein Rechteck zu zeichnen (in unserem Fall gleich 10 Rechtecke).

2) Aus diesen zehn kleinen Rechtecken entsteht letztendlich wieder unsere Diagonale (nur im größeren Maßstab) auf dem Bildschirm (durch die relative Adressierung mit STEP herbeigeführt).

Den gleichen Effekt können wir aber auch mit Hilfe der einseitigen Zuweisung beim LINE-Kommando erreichen (nur Angabe von X2 und Y2):

siehe Kurzlisting LINE 9 im Anhang

Lassen Sie mit Hilfe dieser Programme Ihren Erfindungsreichtum weiterspielen z.B. indem Sie statt eines hochauflösenden Bildschirms eine gerasterte Darstellung aus lauter kleinen Rechtekken entwickeln ...

zu 3: Wir können an das "B", welches für Box steht (siehe unter 'zu 2:') noch einen weiteren Buchstaben anhängen: das F. Dies hat die Bedeutung, daß unser gezeichnetes Rechteck mit Farbe (die gleiche Farbe wie die Rechteckrandfarbe) ausgefüllt wird.

Schauen wir uns diesen Vorgang am besten gleich einmal in unserem 1000-Rechteck-Pogramm von vorhin an:

siehe Kurzlisting LINE 10 im Anhang

Zu Anfang ist das ja sehr beeindruckend, aber schon nach kurzer Zeit scheint sich nichts mehr auf dem Bildschirm zu tun. Dies wirkt nur so, weil andauernd neue Rechtecke entstehen, dieselbigen aber über unsere bereits bestehenden gezeichnet werden ... sprich: wenn dort schon gezeichnet war, stellt unser MSX-Computer die neue Zeichnung halt nicht noch ein zweites Mal dar, denn eine dreidimensionale Wirkung entfalten MSX-Computer und TV-Bildschirm bei weitem noch nicht. Um zu zeigen, daß wahrlich immer neue Rechtecke gezeichnet werden, können wir zu zwei unterschiedlichen Vorgehensweisen greifen:

1) Wir verwenden kleinere Rechtecke

2) Wir verwenden unterschiedliche Farben

1) Ändern Sie hierzu bitte Zeile 50 und 70 wie folgt:

```
50 X2=X1+5
70 Y2=Y2+5
```

Wenn Sie nun unser Programm erneut mit RUN starten, können Sie mitzählen, wie wahrlich tausend Rechtecke auf dem Bildschirm nach und nach entstehen.

Eine andere Möglichkeit, unser Programm für diesen Zweck zu ändern, wäre mit Hilfe des STEP-Unterbefehls im LINE-Kommando. Ändern Sie hierzu bitte Zeile 80 wie folgt:

```
80 LINE(X1,Y1)-STEP(+5,+5),,BF
```

Nun können Sie gar Zeile 50 und Zeile 70 löschen, denn X2 und Y2 errechnen wir uns ja bereits in Zeile 80 automatisch aus den Werten X1 und Y1.

2) Um unterschiedliche Farben für die Rechtecke zu verwenden, geben Sie bitte folgendes Programm ein (oder ändern das vorherige Programm entsprechend ab):

siehe Kurzlisting LINE 11 im Anhang

Nun können wir deutlich erkennen, daß unser MSX-Computer wahrlich in der Lage ist, 1000 verschiedene Rechtecke in beliebiger Größe sehr fix auf den Bildschirm zu bringen. Zu diesem Zweck haben wir Zeile 80 eingefügt, in der nach dem Zufallssystem für jedes zu zeichnende Rechteck eine der 16 zur Verfügung stehenden Farben ausgewählt wird.

Auch hier zeigt sich der MSX-Befehlsschatz mit seinen vielen Unterbefehlen als sehr wirkungsvoll: Wir können Zeile 80 (der COLOR-Befehl) löschen und anstattdessen unseren LINE-Befehl noch ein weiteres Mal um einen Parameter vergrößern:

```
90 LINE(X1,Y1)-(X2,Y2),INT(RND(1)*15),BF
```

Sie werden sicherlich alsbald feststellen, daß es wieder zu Farbvermischungen an manchen Bildschirmpositionen kommt. Wie ich bereits an anderer Stelle in diesem Buch darauf hingewiesen habe, läßt sich dieser klägliche Zustand auf SCREEN 2 leider nicht beheben, denn: die hochauflösende Grafik beträgt zwar 256 * 192 Punkte, die Farbauflösung jedoch "nur" 32 * 192 Punkte. Treffen mehrere Farben eng aufeinander, so ist halt immer die zuletzt aufgetragene Farbe die dominierende, meist über eine ganze Reihe von Bildschirmpunkten hinweg. Doch bedenken Sie: mit etwas Überlegung und Vorsicht können wir mit den zur freien Verfügung stehenden 16 Kilobyte Video-RAM herrlich bunte Bilder mit bis zu 16 Farben auf der hochauflösenden Grafikseite SCREEN 2 erstellen. Bei anderen Computern, die denselben Speicherplatz für die Grafik zur Verfügung stellen, klappt die Darstellung in dieser hohen Auflösung halt nur noch in 4 oder gar in 2 Farben - dann allerdings ohne Gefahr der Farbvermischung.

Wiederum zum Experimentieren für den richtigen Umgang mit der Farbe abschließend zwei kurze Programme:

1) So soll man es nicht machen, denn die unterschiedlichen Farben liegen zu nahe beieinander und führen zu Farbvermischungen auf dem Bildschirm:

siehe Kurzlisting LINE 12 im Anhang

2) In diesem Abstand klappt es immer ohne Farbvermischung, was aber nicht heißen soll, daß unterschiedliche Farben in jedem Fall so weit auseinanderliegen müssen:

siehe Kurzlisting LINE 13 im Anhang

Wollen Sie mehr über das Mißverhältnis zwischen Farbe und hochauflösender Grafik bei den MSX-Computern kennenlernen, schauen Sie bitte unter dem Befehl SCREEN nach und informieren sich damit genauer über den Bildschirmaufbau bei der hochauflösenden Grafik.

LINE 1

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 LINE(128,96)-(150,150)
40 LINE-(160,120)
50 GOTO 50
```

LINE 2

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 DRAW "B M 0,0"
40 LINE(128,96)-(150,150)
50 LINE STEP(10,10)-(180,180)
60 LINE STEP(-20,-20)-STEP(-40,-40)
70 GOTO 70
```

LINE 3

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 FOR N=10 TO 240
40 LINE(128,96)-(N,0)
50 NEXT N
60 GOTO 60
```

LINE 4

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 M=241
40 FOR N=10 TO 240 STEP M
```

```
50 LINE(128,96)-(N,Ø)
60 NEXT N
70 M=M-10
80 IF M<1 THEN GOTO 110
90 CLS
100 GOTO 40
110 GOTO 110
```

LINE 5

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 2
30 LINE(128,96)-(148,96)
40 LINE-(148,76)
50 LINE-(128,76)
60 LINE-(128,96)
70 GOTO 70
```

LINE 6

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 2
30 LINE(128,96)-(148,76)
40 GOTO 40
```

LINE 7

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=1 TO 1000
40 X1=INT(RND(1)*255)
50 X2=INT(RND(1)*255)
60 Y1=INT(RND(1)*191)
70 Y2=INT(RND(1)*191)
80 LINE(X1,Y1)-(X2,Y2),,B
90 NEXT N
100 GOTO 100
```

LINE 8

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 2
30 DRAW "B M 128,96"
40 FOR N=1 TO 10
50 LINE STEP(0,0)-STEP(10,10),,B
60 NEXT N
70 GOTO 70
```

LINE 9

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 2
30 DRAW "B M 128,96"
40 FOR N=1 TO 10
50 LINE-(128+(N*10),96+(N*10)),,B
60 NEXT N
70 GOTO 70
```

LINE 1Ø

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=1 TO 1000
40 X1=INT(RND(1)*255)
50 X2=INT(RND(1)*255)
60 Y1=INT(RND(1)*191)
70 Y2=INT(RND(1)*191)
80 LINE(X1,Y1)-(X2,Y2),,BF
90 NEXT N
100 GOTO 100
```

LINE 11

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=1 TO 1000
40 X1=INT(RND(1)*255)
50 X2=INT(RND(1)*255)
60 Y1=INT(RND(1)*191)
70 Y2=INT(RND(1)*191)
80 COLOR INT(RND(1)*15)
90 LINE(X1,Y1)-(X2,Y2),,BF
100 NEXT N
110 GOTO 110
```

LINE 12

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=100 TO 5 STEP-1
40 LINE(128,96)-STEP(+N,+N),INT(RND(1)*1
5),BF
50 NEXT N
60 GOTO 60
```

LINE 13

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=100 TO 5 STEP-8
40 LINE(128,96)-STEP(+N,+N),INT(RND(1)*1
5),BF
50 NEXT N
60 GOTO 60
```

Befehle Grafik CIRCLE

Zeichnen von Kreisen, Kreisausschnitten und Ellipsen

siehe hierzu auch die Programme im Anhang: CIRCLE 1
CIRCLE 2
CIRCLE 3
CIRCLE 4
CIRCLE 5
CIRCLE 6
CIRCLE 7
CIRCLE 8
CIRCLE 9
CIRCLE 10
CIRCLE 11

Einer der mächtigsten Befehle im MSX-BASIC, der jedoch, so sei gleich zu Ihrer Beruhigung vorangestellt, schon manche Leute mit PAL-Fernsehsystemen gar zur Verzweifelung gebracht hat, denn der Kreis ist gar nicht so rund, wie das eigentlich sein sollte, ist der Befehl CIRCLE:

siehe Kurzlisting CIRCLE 1 im Anhang

Dieser Fehler sei ohne Begründung gleich zu Anfang korrigiert: Geben Sie anstatt der oben abgebildeten Zeile 30 folgendes in Ihren MSX-Computer ein:

```
30 CIRCLE(128,96),50,,,,4/3
```

Wenn Sie nun das Programm noch einmal mit RUN starten ... welch ein Wunder: der Kreis wurde wahrlich rund (es sei denn, Sie haben aus Unkenntnis über diesen Korrekturfaktor Ihren Fernseher in der Vertikalen so dejustiert, daß nun unser eigentlicher Kreis wie eine Pflaume ausschaut; dann ist bei Ihnen allerdings der Kopf des Tagesschau-Sprechers während der Fernsehzeit ebenfalls zur Pflaumenform entstellt - korriegieren Sie also bitte wieder schnell).

Wann können wir den CIRCLE-Befehl verwenden? Nur wenn wir uns auf einem der Grafikbildschirme (SCREEN 2 oder SCREEN 3) befinden, andernfalls gibt's die Fehlermeldung: Illegal function call. Was besagen uns nun die vielfältigen Parameter?

CIRCLE(A,B),C,D,E,F,G

A und B geben den Kreismittelpunkt an. Diesen können wir uns einerseits dorthin wünschen, wo er im Rahmen der 256*192-Matrix erscheinen soll (absolute Eingabe) oder aber um so und so viele Punkte vom zuletzt eingegebenen Punkt entfernt (mit Hilfe des Vorparameters STEP = relative Eingabe). Für beide Möglichkeiten je ein Beispiel:

1) absolute Eingabe:

siehe Kurzlisting CIRCLE 2 im Anhang

2) relative Eingabe:

siehe Kurzlisting CIRCLE 3 im Anhang

Während wir bei der absoluten Eingabe die gewünschten Koordinaten für den Kreismittelpunkt genau angeben müssen, können wir bei der relativen Adressierung mit Hilfe des STEP-Parameters von dem letzten angewählten Punkt (in CIRCLE 3 ist das 128,96) als dem neuen Nullpunkt ausgehen (von dort aus wird die CIRCLE-Eingabe berechnet).

C gibt den Radius des gewünschten Kreises an, wobei wir nicht unbedingt im Bildschirmrahmen bleiben müssen (die Radiusangabe darf so groß oder so klein sein, wie dies bei Verwendung einer Integerzahl nun einmal möglich ist (zwischen +32767 und -32768). Geben Sie einen noch größeren oder noch kleineren Wert für C ein, zeigt Ihnen Ihr MSX-Computer einen 'Overflow error' an).

D gibt die Farbnummer an, in der der Kreis gemalt werden soll. Hierbei sollten Sie bei den MSX-Computern nicht beliebig verfahren, da es sonst gleich zu unschönen Flächeneinfärbungen kommen kann (da die Auflösung wohl 256*192 Punkte, die Farbauflösung

aber lediglich 32*192 Punkte beträgt). Das folgende Programm zeigt Ihnen (leider) eindeutig das unschöne Ineinanderlaufen verschiedener Farben:

siehe Kurzlisting CIRCLE 4 im Anhang

Diesen unschönen Effekt können Sie am leichtesten umgehen, indem Sie einen größeren Abstand zwischen den Kreisen wählen z.B. ändern Sie Zeile 30 in:

```
30 FOR N=1 TO 200 STEP 8
```

E und F sind zwar wunderschöne Parameter, um Kreisanfang und Kreisende zu bestimmen (so lassen sich z.B. Kuchendiagramme verhältnismäßig einfach erstellen), aber man muß sie auch richtig anzuwenden wissen. Das Problem liegt dabei in der Verarbeitung des MSX-Computers: Die Eingabe wird nicht in Grad gefordert, sondern in sogenannten Radian-Angaben.

Zudem muß man sich - wohl oder übel - daran gewöhnen, daß das MSX-BASIC einen Kreis nicht oben (bei 0 Grad), sondern erst in der Drei-Uhr-Stellung (bei 90 Grad) zu zeichnen beginnt und zwar in der Richtung entgegen dem Uhrzeigersinn. Noch ein Hinweis, bevor wir die Arbeit mit Radian in kleinen Beispielen anzuwenden kennenlernen: Die Angabe für Kreisanfang und Kreisende muß sich im Bereich von +2 PI (+6.21318) bis -2 PI (-6.21318) bewegen, andere Werte werden mit der Fehlermeldung 'Illegal function call' quittiert.

Zeichnen wir zuerst einen Halbkreis (180 Grad) von der Drei-Uhr-Position angefangen (180 Grad bedeuten in Radian 1 PI = 3.14159):

siehe Kurzlisting CIRCLE 5 im Anhang

Zeichnen wir als weiteres einen Viertelkeis von der Neun-Uhr-Position (180 Grad) angefangen:

siehe Kurzlisting CIRCLE 6 im Anhang

Wie sähe gar ein gestrichelter Kreis aus? Schauen wir uns das Beispiel an:

siehe Kurzlisting CIRCLE 7 im Anhang

Wenn Sie schon ein klein wenig Kenntnis vom DRAW-Befehl bzw. der sogenannten Macro-Graphic-Language (MGL) haben, werden Sie nun auch verstehen, wie wir ein einfaches viergeteiltes Kuchendiagramm zeichnen können:

siehe Kurzlisting CIRCLE 8 im Anhang

Nun noch, bevor ich Ihnen den Trick verrate, wie Sie Radians einfach in Grad (und natürlich auch umgekehrt) umrechnen können, noch die grafische Darstellung einer biologische Form, die die Natur im Laufe von Jahrmillionen entwickelt hat (vergleichen Sie einmal die innere weiße Fläche mit der Form eines Schneckenhauses) und die mit Ihrem MSX-Computer innerhalb von Sekunden nachentwickelt wird:

siehe Kurzlisting CIRCLE 9 im Anhang

Schön, nicht wahr?

Nun aber zur Umrechnungsmethode, denn wozu haben wir schließlich unseren leistungsfähigen MSX-Computer? Auch zum Rechnen!

Wir teilen einfach unsere gewünschte Gradangabe durch den Wert 57.2958 (dieser Wert entsteht durch die Umrechnung von 360 Grad durch 2 * PI: 360/6.38218 = 57.2958):

180 Grad sind dann PI oder 3.14159
60 Grad sind dann 1.047197 usw.

Wir können dies in unseren Programmen mit Hile einer Funktionszuweisung festlegen:

```
10 DEF FNA(GRAD)=GRAD/57.2958
20 INPUT "Grad eingeben ";GRAD
30 PRINT FNA(GRAD)
```

Nun noch zum letzten Parameter im CIRCLE-Befehl, den wir ja bereits zu Beginn kennen- und schätzengelernt haben:

G die Verzerrung des Kreises hin zur Ellipse. Die Werte für G dürfen zwischen 1/260 und 260 liegen, jedoch ist eine Verzerrung des Kreises bis hin zur Linie eher uninteressant; deshalb werden normalerweise Verzerrungen zwischen 1 (Normalstatus) und 0 (der Kreis verzieht sich in der X-Ausdehnung) oder aber zwischen 1 und 2 (der Kreis verzieht sich in der Y-Ausdehnung) verwendet.

Um diesen Effekt zum Abschluß der CIRCLE-Behandlung noch einmal im Bilde zu verdeutlichen, können Sie folgendes kurzes Prgramm eingeben:

siehe Kurzlisting CIRCLE 10 im Anhang

Noch ein paar Hinweis zum Schluß: Es dürte aus den Beispielprogrammen ersichtlich geworden sein, daß wir nicht alle Parameter im CIRCLE-Befehl jedesmal verwenden müssen; wir sind lediglich dazu gezwungen, in jedem Fall bei Verwendung des CIRCLE-Befehls Kreismittelpunkt (A,B) und Radius (C) anzugeben. Die zuletzt verwendete Vordergrundfarbe wird übernommen, zu überspringende Parameter kennzeichnen wir durch Kommata.

Da Ihnen die Umrechnung von Grad in Radian nun keine Schwierigkeit mehr bereitet, werden Sie auch die Variablenzuweisung

```
A=57.2958
```

im folgenden Programm (Zeile 50) verstehen, ohne daß wir hier auf den Ursprung dieser Zahl noch einmal genauer eingehen müssen:

In Programm CIRCLE 11 arbeiten wir mit einer Folge von BASIC-Kommandos, die Werte suksessive errechnen, um mit PSET nach Ihren Vorgaben für Radius, X-Y-Koordinaten und sogar Punktabstand einen Kreis zu erstellen. Zum Vergleich habe ich auch einen CIRCLE-Befehl mit kleinerem Radius und einer anderen Vordergrundfarbe in das Programm eingebaut, so daß Sie deutlich sehen können, wie schnell das Kreiszeichnen (in Maschinensprache) mit dem Befehl CIRCLE und wie langsam dies mit unserem BASIC-Programm gelingt:

siehe Kurzlisting CIRCLE 11 im Anhang

Dafür kann dieser BASIC-CIRCLE-Befehl aber auch zumindest an einer Stelle mehr als das Maschinensprach-CIRCLE-Kommando: Sie können einen zusätzlichen Parameter eingeben, der den Punktabstand beim Kreiszeichnen kontrolliert.

CIRCLE 1

```
        -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 CIRCLE(128,96),50
40 GOTO 40
```

CIRCLE 2

```
        -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 CIRCLE(128,96),20
40 CIRCLE(64,80),20
50 GOTO 50
```

CIRCLE 3

```
        -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 PSET(128,96)
40 CIRCLE STEP(-20,0),30
50 CIRCLE STEP(+60,30),30
60 GOTO 60
```

CIRCLE 4

```
        -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 FOR N=1 TO 100
40 CIRCLE(128,96),N,INT(RND(1)*15)
50 NEXT N
60 GOTO 60
```

CIRCLE 5

```
        -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 CIRCLE(128,96),30,,0,3.14159
40 GOTO 40
```

CIRCLE 6

```
        -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 CIRCLE(128,96),30,,3.14159,4.712385#
40 GOTO 40
```

CIRCLE 7

```
        -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 FOR N=0 TO 6.21318 STEP 1.55795
40 CIRCLE(128,96),30,,N,N+1
50 NEXT N
60 GOTO 60
```

CIRCLE 8

```
        -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 FOR N=0 TO 4.712385# STEP 1.55995
40 CIRCLE(128,96),30,,N,N+1.55795
50 Z=Z+1
```

```
60 IF Z=1 THEN DRAW "N u30"
70 IF Z=2 THEN DRAW "N l30"
80 IF Z=3 THEN DRAW "N d30"
90 IF Z=4 THEN DRAW "N r30"
100 FOR M=1 TO 100:NEXT M
110 NEXT N
120 GOTO 120
```

CIRCLE 9

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 2
30 Z=1
40 FOR N=0 TO 6.28318 STEP .1
50 Z=Z+1
60 CIRCLE(128,96),Z,,0,N
70 NEXT N
80 GOTO 80
```

CIRCLE 10

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 2
30 FOR N=2 TO 1/260 STEP -.1
40 CIRCLE(128,96),60,,,,N
50 NEXT N
60 GOTO 60
70 NEXT N
80 GOTO 80
```

CIRCLE 11

-*-*-*-*-*-

```
10 COLOR 15,4
20 CLS
30 INPUT "Radius, X, Y, Punktabstand (R,
X,Y,P) ";R,X,Y,P
40 SCREEN2
50 A=57.2958
60 CIRCLE(X,Y),R-10
70 FOR B=1 TO 360 STEP P
80 X1=R*COS(B/A)+X
90 Y1=R*SIN(B/A)+Y
100 PSET(X1,Y1),1
110 NEXT B
120 GOTO120
```

Befehle Grafik PAINT

Ausmalen von umrandeten Flächen auf den Grafikbildschirmen

siehe hierzu auch die Programme im Anhang: PAINT 1
PAINT 2
PAINT 3
PAINT 4

Im Gegensaz zu verschiedenen anderen BASIC-Dialekten verfügt das MSX-BASIC nicht nur über die Grafikefehle zum Setzen eines Punktes (PSET) bzw. Zeichnen einer Linie (LINE), Sie können vielmehr auch umschlossene Felder auf dem Bildschirm darstellen (LINE ,,B) und dieselbigen mit PAINT ausmalen.

Schauen wir uns einmal an, wie leicht verschiedene zu umschließende Felder mit MSX-Grafikbfehlen auf den Bildschirm dargestellt werden können:

siehe Kurzlisting PAINT 1 im Anhang

In diesem kurzen Programm sehen wir gleich vier verschiedene Möglichkeiten, wie wir fest umschlossene grafische Gebilde erstellen können: Ganz innen ein Kreis (CIRCLE - Zeile 30), als nächstes ein Rechteck (LINE ,,B) ebenfalls mit nur einem Befehl erstellt (Zeile 40), schließlich ein noch größeres Rechteck (DRAW - gezeichnet mit Hilfe unserer luxuriösen Grafikprogrammiersprache in Zeile 50) und letztendlich ein noch größeres Rechteck auf den Bildschirm gebracht durch mehrere Linien im 90-Grad-Winkel zueinander (LINE - Zeilen 60 bis 90).

Doch wir wollen uns bei der Erklärung des PAINT-Befehls nicht mit Gitterbauten begnügen, wir benötigen sie nur, um diese Flächen auszumalen: Für diesen Zweck wenden wir den PAINT=Ausmalbefehl an: Wir suchen uns einen Punkt innerhalb (!) des auszumalenden Gebildes und wenden anschließend den PAINT-Befehl an. Zum Ausmalen des Kreises brauchen wir keine Rechenkünstler zu sein, denn der Kreismittelpunkt (in unserem Programm Koordi-

nate 128,96) liegt innerhalb des Kreises. Da wir zur Zeit noch nicht wissen, wie man mit PAINT verschiedene Farben zum Ausmalen heranzieht, begnügen wir uns mit der gleichen Farbe, mit der auch die Gittergrundzeichnung entstanden ist. Ähnlich arbeiten wir uns Koordinate für Koordinate, umschlossenen Körper für umschlossenen Körper voran, bis folgende zusätzliche Befehle für das Ausmalen zusammenkommen:

1) PAINT(128,96) für den Kreis
2) PAINT(85,105) für das LINE-B-Rechteck
3) PAINT(75,115) für das DRAW-Rechteck
4) PAINT(65,125) für das viergeteilte LINE-Rechteck

siehe Kurzlisting PAINT 1a im Anhang

Wie bei anderen MSX-Grafikbefehlen (LINE, PSET) sind wir auch bei PAINT nicht dazu gezwungen, absolut die Koordinaten anzugeben (z.B. 128,96), von wo mit dem Ausmalen begonnen werden soll. Vielmehr können wir auch relativ den Ausmalpunkt vom Computer errechnen lassen. Bei unserem Beispiel lohnt sich dies jedoch lediglich, wenn es darum geht, den Kreis in Zeile 20 mit Farbe zu füllen: Der zuletzt angesteuerte Grafikpunkt ist hierbei der Kreismittelpunkt = 128,96. Soll auch an dieser Stelle mit dem Ausmalen begonnen werden, geben wir in Zeile 25 ein:

```
35 PAINT STEP (0,0)
```

d.h.: gehe vom Kreismittelpunkt (dem zur Zeit aktuellen Grafikcursorpunkt) 0 Punkte nach rechts und 0 Punkte nach unten - schon bist Du - logisch - wieder bei Punkt 128,96 ... und hier beginne mit dem Ausmalen mit der zur Zeit aktuellen Farbe.

Wie bereits vorhin kurz darauf hingewiesen, malt der PAINT-Befehl im Normalfall nur in der Farbe aus, die er gleichzeitig als Rahmenbegrenzung vorfindet. Um dies zu sehen, ändern Sie an beliebiger Stelle vor (!) einem PAINT-Befehl mit COLOR die Farbe. Verheerendes passiert, denn der gesamte Bildschirm wird nun z.B. durch

```
33 COLOR 3
```

Mit dieser Farbe gefüllt und unsere vorherige Zeichnung ist übermalt bzw. gelöscht. Dies hängt damit zusammen, daß Ihr MSX-Computer beim Ausmalen weiterarbeitet, bis er die gleichen Farben als Randbegrenzung wieder vorfindet. Da wir bisher auf dem Bildschirm nicht mit Farbe 3 gezeichnet haben, wird alles ausgemalt.

Möglich ist es hingegen, mit verschiedenen Ausmalfarben zu arbeiten, wenn wir entsprechend auch die Umrandungsfarbe ändern. Zu diesem Zweck geben Sie bitte folgendes Programm in Ihren MSX-Computer ein:

siehe Kurzlisting PAINT 2 im Anhang

Nach und nach werden Kreise mit jeweils unterschiedlicher Farbe gezeichnet und mit der gleichen Farbe auch ausgemalt. Wie Sie allerdings sehen können, entstehen dabei (wieder einmal!!!) die Farbüberlagerubngen. Gehen Sie deshalb bitte ein bißchen vorsichtig mit so vielen Farben in derart geringem Abstand auf einer Zeichnung um (SCREEN 2), denn die MSX-Computer akzeptiern halt nur einmal 2 verschiedene Farben in jedem 8*1-Punktmuster.

In unserem obigen Programm können wir auf den COLOR-Befehl in Zeile 30 vollständig verzichten, indem wir die Farbinformation sowohl dem CIRCLE- als auch den PAINT-Befehl mitteilen; zu dieser Vorgehensweise siehe unter CIRCLE und im weiteren Text.

Speziell auf den PAINT-Befehl bezogen besitzt derselbige wie so viele andere MSX-Befehle ebenfalls Parametereingabemöglichkeiten, wobei Parameter Nummer 1 angibt, mit welcher Farbe ausgemalt werden soll. Ändern Sie also obiges Programm folgendermaßen:

siehe Kurzlisting 2a im Anhang

Alles läuft wie vorhin ab nur halt mit dem Unterschied, daß wir auf die COLOR-Anweisung verzichtet haben (das ist auch manchmal gut so, denn die COLOR-Anweisung bezieht sich ja leider auch auf die gesamten Textbildschirme SCREEN 0 und SCREEN 1!). Wir können aber auch sowohl mit einer gesondert angegebenen Ausmalfarbe (Parameter 1) als auch einer gesondert angegebenen Umrandungsfarbe (Parameter 2) - letztere Farbe ist diejenige, die den Rand unserer auszumalenden Fläche kennzeichnet - arbeiten (dieser Parameter funktioniert jedoch nur auf SCREEN 3):

PAINT X, Y, Ausmalfarbe, Umrandungsfarbe

Wir malen uns zur Demonstration drei Kreise auf den Bildschirm und lassen anschließend den innersten Kreis mit der Farbe des äußersten Kreises (Rand) ausmalen:

siehe Kurzlisting PAINT 3 im Anhang

Ärgern Sie sich bitte nicht über die Farbüberschreitungen (eine Art Ineinanderlaufen der Farben): das liegt, wie schon so oft in diesem Buch gesagt, an der begrenzten Farbauflösung der MSX-Computer in der hochauflösenden Grafik auf SCREEN 2.

Noch ein paar Hinweise:

1) Wir haben 2 Befehle, um Flächen auf dem Grafikbildschirm auszumalen: PAINT und LINE(...)-(...),,BF. Während der LINE-BOX-FILL-Befehl nur rechteckige Flächen ausmalt, klappt mit PAINT das Ausmalen bis in jede Ecke. Wollen Sie größere Flächen einfärben, verwenden Sie bitte vorwiegend den LINE-BF-Befehl. Geschwindigkeitsbeweise werden Sie davon überzeugen:

```
1) 10 SCREEN 2
   20 PAINT(128,96),1
   30 GOTO 30

   15 Sekunden
```

```
2) 10 SCREEN 2
   20 LINE (0,0)-(255,191),1,BF
   30 GOTO 30
```

1 Sekunde

2) Der PAINT-Befehl schaut nur in 4 Richtungen (oben, unten, links und rechts), ob dort ein Rand zu erkennen ist. Über diagonale Begrenzungen kann er deshalb leicht hinausschießen und, wie bereits vorgeführt, unser ganzes Bild auf einmal zerstören.

Im folgenden kleinen BASIC-Programm zeigen wir, wie der PAINT-Befehl arbeitet, ohne auf die einzelnen Grafikbefehle näher eingehen zu wollen:

siehe Kurzlisting PAINT 4 im Anhang

Hier fragen wir ab, ob die gewünschte Umrandungsfarbe schon erreicht ist. Die Geschwindigkeit ist allerdings bei diesem BASIC-Programm noch einmal langsamer im Vergleich zum PAINT-Befehl. Das Programm ist zudem sehr viel langsamer im Vergleich zum LINE-BF-Befehl.

Noch zwei Tips für Speicher-PEEK-POKE-Fanatiker: Der Bearbeitungsspeicher des PAINT-Befehls beginnt im RAM an Speicherstelle &HF949. Die Farbe, die PAINT als Randfarbe akzeptiert, ist bei &HFCB2 abgespeichert.

PAINT 1

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 CIRCLE(128,96),10
40 LINE(80,110)-(150,70),,B
50 DRAW " B M 70,120 R90 U60 L90 D60"
60 LINE(60,130)-(170,130)
70 LINE -(170,50)
80 LINE -(60,50)
90 LINE -(60,130)
100 GOTO 100
```

PAINT 1a

```
-*-*-*-*-*-
35 PAINT(128,96)
45 PAINT(85,105)
55 PAINT(75,115)
95 PAINT(65,125)
```

PAINT 2

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 FOR N=1 TO 14
40 COLOR N
50 CIRCLE(128,96),50
60 PAINT STEP(0,0)
70 NEXT N
80 GOTO 80
```

PAINT 2a

```
-*-*-*-*-*-
40 REM
50 CIRCLE(128,96),50,N
60 PAINT STEP(0,0),N
```

PAINT 3

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 3
30 FOR N=1 TO 3
40 CIRCLE(128,96),N*10,N
50 NEXT N
60 PAINT(128,96),3,1
70 GOTO 70
```

PAINT 4

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 LINE(128,96)-(158,46),,B
40 FOR N=129 TO 157
50 FOR M=95 TO 47 STEP-1
60 IF POINT(N,M)<>1 THEN PSET(N,M)
70 NEXT M
80 NEXT N
90 GOTO 90
```

Dimensionierte Variable, die das Aussehen der Sprites enthält

siehe hierzu auch die Programme im Anhang: SPRITE$(N) 1
SPRITE$(N) 2
SPRITE$(N) 3
SPRITE$(N) 4
Programm 1

Die meisten Grafikbefehle können Sie nur auf den eigentlichen Grafikbildschirmen SCREEN 2 und SCREEN 3 anwenden. Versuchen Sie, Grafikbefehle auf SCREEN 0 oder SCREEN 1 einzugeben, erfolgt die Fehlermeldung 'Illegal function call'. Bei SPRITE$(N) ist dies, obwohl es sich bei diesem Kommando auch um einen Grafikbefehl handelt, ein klein wenig anders: Es erfolgt keine Fehlermeldung, wenn Sie diesen Befehl auf SCREEN 1 anwenden (auf SCREEN 1 lassen sich ja auch wohlgemerkt Sprites zur Darstellung bringen). Die Fehlermeldung wird auch auf SCREEN 0 nicht ausgegeben, allerdings können Sie die damit erzeugten Spritemuster auch nicht zur Anzeige bringen. Wofür ist dann der Befehl SPRITE$ auf SCREEN 0 sinnvoll? Sie verfügen mit diesem Befehl über eine dimensionierte Variablenzuweisung, die allerdings keinen Speicherplatz im normalen RAM, sondern im VRAM einnimmt. Arbeiten Sie also nur auf SCREEN 0, können Sie mit SPRITE$ Zeichenketten mit jeweils bis zu 32 Zeichen belegen, ohne vorher mit CLEAR Speicherraum dafür zu reservieren oder mit DIM eine Variablendimensionierung vorzunehmen. Mit dem zweiten SCREEN-Parameter können Sie die Anzahl der zu verwendenden SPRITE$-Variablen vorgeben:

SCREEN ,0 und SCREEN ,1:
256 Variablen mit bis zu jeweils 8 Zeichen
SCREEN ,2 und SCREEN ,3:
64 Variablen mit bis zu jeweils 32 Zeichen

Soviel vorweg. Selbstverständlich können Sie diese etwas überraschende Verwendung des SPRITE$(N)-Befehls (eigentlich handelt es sich hierbei vielmehr um eine Variablenzuweisung) auch in den SCREENS 1 bis 3 vornehmen, jedoch können Sie dann die vollständige Spriteanwendung nicht mehr in Betracht ziehen (Sie können allerdings bei Verwendung von nur 8 Sprites den VRAM-Speicherplatz der restlichen Sprites zur Variablenabspeicherung verwenden; in jedem Fall gelten die oben angegebenen Zahlwerte als Richtwerte.

Nun aber zur Bedeutung des SPRITE$(N)-Befehls im eigentlichen: Sie geben mit SPRITE$(N) an, wie das N-te Sprite auszusehen hat. Da bei einem 8*8-Sprite=64 einzelne Punkte nur 8 Zeichen für das Aussehen verantwortlich sind, ist die Spritedefinition mit direkter Eingabe von der Tastatur gar nicht so einfach. Wie funktioniert die Darstellung einer Spritelinie (8 gesetzte bzw. nicht gesetzte Bildschirmpunkte) durch ein Zeichen nun im einzelnen? Dazu müssen Sie wissen, daß Ihr Computer jeden Buchstaben und jedes Zeichen gar nicht als ein solches erkennt, sondern vielmehr dahinter eine Zahl zwischen 0 und 255 sieht. Wenn Sie darüber näheres erfahren wollen, schauen Sie bitte unter der Befehlserläuterung von CHR$ bzw. ASC nach. Hier nur soviel: Der Buchstabe 'A' bedeutet für Ihren MSX-Computer die Zahl 65.

Wir müssen nun noch eine Stufe weiter zurückgehen, um hinter dieser '65' acht gesetzte oder auch nicht gesetzte Bildschirmpunkte zu erkennen. Zu diesem Zweck müssen wir die Darstellung der Zahl 65 in Computerschrift verstehen: Ihr MSX-Computer rechnet sich das uns vertraute Zehner- oder Dezimalsystem in das Zweier- oder Binärsystem um. Als Ergebnis unser 65 bekommen wir in Binärdarstellung die Zahl 1000001. Probieren Sie es selbst einmal aus mit dem MSX-BASIC-Befehl BIN$:

```
PRINT BIN$(65)
```

Da unsere Binärzahl für die 8-Punkt-Darstellung allerdings über 8 Nullen für nicht gesetzte Punkte und Einsen für gesetzte Punkte verfügen muß, heißt es, die vorderen Stellen mit Nullen zu ergänzen:

```
PRINT STRING$(8-LEN(BIN$(65)),"0")+BIN$(65)
```

Nun können Sie sehen, wie die erste Zahl unseres Sprites ausschaut, wenn die SPRITE$-Zuweisung mit 'A' beginnt. Also können wir bei einer 8*8-Sprite-Matrix mit nur 8 Zeichen z.B. achtmal den Buchstaben A ein Punktmuster von 8*8=64 Einzelpunken oder An-Aus-Zuständen definieren.

Der normale Weg der Spritedefinition geht aber nicht nach dem Trial and Error System vor sich, das wir gerade eben kennengelernt haben. Vielmehr entwerfen Sie das Spriteaussehen vorher auf einem Papier und erst anschließend wird das gewünschte Zeichen (z.B. ein 'A') errechnet. Zu diesem Zweck müssen Sie einerseits die Binärdarstellung (mit Nullen und Einsen) lesen und anschließend in ein Zeichen übertragen können. Schauen wir uns dazu das folgende Programm an:

siehe Kurzlisting SPRITE$(N) 1 im Anhang

Wie ist die Spritedefinition vor sich gegangen?

1) In den Zeilen 90 bis 160 ist das Sprite in 64 Einzelpunkten (= Nullen für nicht gesetzte Punkte bzw. Einsen für gesetzte Punkte) dargestellt worden d.h. immer dort, wo eine 1 geschrieben steht, wird später im Sprite ein Punkt gesetzt sein, wo eine 0 steht, bleibt die entsprechende Position im Sprite leer.

2) Nach und nach (Zeile 30 bis 70) werden die 8 Spritelinien eingelesen (Zeile 40), mit dem Binärschlüssel ergänzt (Zeile 50) und schließlich als Charakterzeichen umgewandelt und zu unserem Sprite nach und nach hinzugezählt (Zeile 60).

Auf diese erst einmal kompliziert anmutende Art und Weise entsteht unser Sprite und sieht schließlich im Charakteraussehen gar nicht mehr so aus, wie wir es ursprünglich gemalt hatten:

SPRITE$(1)="AUAUAUAU"

Nun sollte es Ihnen aber deutlich geworden sein, wie mit Hilfe dieser platzsparenden Definition das Sprite entwickelt wurde. Schauen wir uns zur Überprüfung unserer Spritedefinition einmal das Ergebnis auf dem Bildschirm an: Dafür müssen wir unser soeben geschriebenes Programm ein klein wenig erweitern:

siehe Kurzlisting SPRITE$(N) 1a im Anhang

Sie merken an der Geschwindigkeit Ihres MSX-Computers bei der Umwandlung der 8 Zeichen in die nun auf dem Bildschirm dargestellte Punktvielfalt, daß die Computersprache alle Rechnungen mit Nullen und Einsen ganz einfach und schnell meistert.

Wie bereits oben vermerkt, können Sie in der 8*8-Darstellung bis zu 256 Spriteaussehen im VRAM ablegen. Jeweils mit PUTSPRITE und der entsprechenden Spritenummer steht es Ihnen anschließend frei, das gerade benötigte Sprite auch auf dem Bildschirm erscheinen zu lassen.

Wie Ihnen aus dem SCREEN-Befehl bekannt sein wird, können Sie die 8*8-Spritegröße mit Hilfe des zweiten SCREEN-Parameters vorbestimmen:

SCREEN,0

Es ist aber auch möglich, bei der 8*8-Spritedefinition

SCREEN,1

anzuwählen. Auf dem Bildschirm erscheint dann zwar eine 16*16-Spritedarstellung, jedoch wird nur jeder Punkt als 2*2-Großpunktdarstellung verwendet.

Schauen wir uns den Unterschied von SCREEN,1 und SCREEN,2 gleichfalls durch Erweiterung unseres vorherigen Spriteprogramms etwas genauer an:

siehe Kurzlisting SPRITE$(N) 1b im Anhang

Es ist auch möglich, 16*16-Sprites darzustellen, die nicht nur aus 8*8 vergrößerten Punkten zusammengesetzt sind, sondern wahrlich aus 16*16 eigenständigen Punkten bestehen. Dafür müssen wir

SCREEN,2 bzw.
SCREEN,3

(letzterer wieder für die vergrößerte 16*16-Darstellung) anwählen. In der Spritedefinition muß bei dieser Größenordnung etwas differenzierter vorgegangen werden: Sie müssen sich zu diesem Zweck darüber im klaren sein, daß ein 16*16-Sprite eigentlich aus 4 Sprites jeweils mit dem Muster 8*8 zusammengesezt ist. Bei der Spritedefinition besteht in dieser Größenordnung also ein Sprite nicht mehr als lediglich aus 8 Zeichen, sondern vielmehr aus 4*8 Zeichen = 32 Zeichen. Die Zusammensetzung geht dann so vor sich:

Erst wird das 8*8-Sprite links oben gelesen, dann das 8*8-Sprite links unten, anschließend rechts oben und schließlich das Sprite rechts unten.

Unser Lesevorgang verkompliziert sich dadurch nun ein wenig:

siehe Kurzlisting SPRITE$(N) 2 im Anhang

Damit wir nicht nur die fertige Zeichendastellung nach unserer langen Programmeingabe betrachten können, wollen wir die Vorgehensweise zur Spritedefinition ein klein wenig genauer untersuchen:

Zur normalen 16*16-Darstellung müssen wir das Programms um SCREEN ,2 und PUTSPRITE erweitern.

Wollen wir unser 16*16-Sprite auch noch vergrößert darstellen, geben wir das Kommando SCREEN,3 zu Beginn des Programms ein.

Es ist gar auch möglich, unser Riesensprite nur in einer 8*8-Matrix in SCREEN,0 bzw. SCREEN,1 darzustellen. Zu diesem Zweck geben Sie bitte zu Beginn des Programms

SCREEN ,1

bzw. für vergrößerte 8*8-Darstellung

SCREEN ,2

ein.

Da allerdings nur 8*8 Punkte in den zwei zuletzt angewendeten Modi dargestellt werden können, sehen Sie halt nur einen Ausschnitt d.h. die linke obere Ecke Ihres eigentlich definierten 16*16-Sprites d.h. das vorher definierte 16*16-Sprite wird automatisch auf das erste 8*8-Spritemuster links oben gekürzt.

Schauen wir uns nun einmal an, wie die Spritelesung bei einem 16*16-Sprite abläuft: Zuerst werden sämtliche Spriteaussehen - Informationen in eine vorher dimensionierte Stringvariable eingelesen (Zeile 40). Danach werden sämtliche ersten Spritehälften interpretiert und zur Spritedefinition herangezogen. Anschließend geschieht das gleiche mit den 8 rechten Spritedefinitionsziffern (dezimal gerechnet). So entsteht schließlich unser 16*16-Sprite - eigentlich zusammengesetzt aus 4 Sprites jeweils mit der Punktmatrix 8*8.

Sie können im VRAM bis zu 64 der großen 16*16-Sprites ablegen (im Vergleich zu 256 Sprites der Größe 8*8), jedoch sind auf dem Bildschirm gleichzeitig 'nur' 32 Sprites darstellbar. Treffen Sie also immer die richtige Auswahl.

Nun noch ein paar interessante Hinweise:

1) Sie können im VRAM nachlesen, wo das Spriteaussehen abgelegt ist: Zu diesem Zweck bedienen Sie sich der Hinweisvariablen BASE(8) für SCREEN 1 bzw. BASE(13) für SCREEN 2 bzw. BASE(18) für SCREEN 3 und zählen 8 hinzu, um jedes Spriteaussehen abzurufen z.B. um die 8 Zahlen der Zeilen bzw. das binäre Aussehen von Sprite 1 in SCREEN 1 zu sehen, geben Sie folgendes ein:

siehe Kurzlisting SPRITE$(N) 3 im Anhang

Bei der Abbildung eines 16*16-Sprites direkt durch Abfrage des VRAM ist das etwas komplizierter. Schauen wir uns hierzu das Vorgehen der Binärumrechnung ein klein wenig genauer an:

siehe Kurzlisting SPRITE$(N) 4 im Anhang

2) Seien Sie sehr vorsichtig, wenn Sie zwischendurch im Programm die Spritegröße verändern wollen. Jede SCREEN-Änderung zum Zwekke des SCREENwechsels (Anwahl des ersten SCREEN-Parameters) bzw. der Spritegröße (Anwahl des zweiten SCREEN-Parameters) löscht das gesamte Spriteaussehen im VRAM. Es geht auch anders, aber dazu müssen Sie noch sehr viel tiefer in Ihren MSX-Computer einsteigen.

siehe Programm 1 im Anhang

3) Sie können jederzeit abfragen, wie viel Spritespecherplatz (Angabe in Anzahl der Zeichen) noch frei ist:

```
PRINT SPRITE$()
```

Arbeiten Sie mit der 8*8-Spritematrix, brauchen Sie die gerade ausgegebene Zahl nur durch 8 zu teilen, das Ergebnis ist die Anzahl der noch nicht definierten Spritemuster. Verwenden Sie hingegen die 16*16-Spritedarstellung, teilen Sie bitte den mit PRINT SPRITE$() ausgegebenen Wert zur Vermeldung des freien Speicherplatzes durch 16.

SPRITE$(N) 1 -*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 1
30 FOR N=1 TO 8
40 READ A$
50 A$="&B"+A$
60 B$=B$+CHR$(VAL(A$))
70 NEXT N
80 SPRITE$(1)=B$:PRINT SPRITE$(1)
90  DATA 01000001
100 DATA 01010101
110 DATA 01000001
120 DATA 01010101
130 DATA 01000001
140 DATA 01010101
150 DATA 01000001
160 DATA 01010101
```

SPRITE$(N) 1a -*-*-*-*-*-

```
85 PUT SPRITE 1,(128,96)
```

SPRITE$(N) 1b -*-*-*-*-*-

```
15 IF Z=1 THEN Z=0 ELSE Z=1
20 SCREEN 1,Z
25 RESTORE
86 LOCATE 0,0
87 PRINT "Spritegroesse";Z
88 FOR O=1 TO 1000
89 NEXT O:GOTO 15
```

SPRITE$(N) 2 -*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 1
30 DIM A$(16)
40 FOR N=1 TO 16
50 READ A$(N)
60 NEXT N
70 FOR N=1 TO 16
80 A$=A$+CHR$(VAL("&B"+LEFT$(A$(N),8)))
90 NEXT N
100 FOR N=1 TO 16
110 A$=A$+CHR$(VAL("&B"+RIGHT$(A$(N),8))
)
120 NEXT N
130 SPRITE$(1)=A$
140 DATA 0101010101010101
150 DATA 1010101010101010
160 DATA 0101010101010101
170 DATA 1010101010101010
180 DATA 0101010101010101
```

```
190 DATA 1010101010101010
200 DATA 0101010101010101
210 DATA 1010101010101010
220 DATA 0101010101010101
230 DATA 1010101010101010
240 DATA 0101010101010101
250 DATA 1010101010101010
260 DATA 0101010101010101
270 DATA 1010101010101010
280 DATA 0101010101010101
290 DATA 1010101010101010
```

SPRITE$(N) 3

-*-*-*-*-*-

```
10 FOR N=14336+32 TO 14336+39
20 PRINT STRING$(8-LEN(BIN$(VPEEK(N))),"
0")+BIN$(VPEEK(N))
30 NEXT N
```

SPRITE$(N) 4

-*-*-*-*-*-

```
10 FOR N=14336+32 TO 14336+47
20 PRINT STRING$(8-LEN(BIN$(VPEEK(N))),"
0")+BIN$(VPEEK(N));
30 PRINT STRING$(8-LEN(BIN$(VPEEK(N+16))
),"0")+BIN$(VPEEK(N+16))
40 NEXT N
```

Befehle Grafik PUTSPRITE

Die Attribute (Farbe, Position usw.) der Sprites verändern

siehe hierzu auch die Programme im Anhang: PUTSPRITE 1
MISTER MINER
FRETY
Programm 4
Programm 5

Mit unserem MSX-Computer haben wir die Möglichkeit, bis zu 256 Sprites im Video-RAM-Speicher mit unterschiedlichem Aussehen abzulegen sowie bis zu 32 Sprites gleichzeitig auf dem Bildschirm erscheinen zu lassen.

Bei der Spritedefinition bedienen wir uns der Variablenzuweisung mit dem Kommando SPRITE$, beim Abruf der Sprites auf den Bildschirm zu einer bestimmten Koordinate gebrauchen wir hingegen das Kommando PUTSPRITE, was soviel bedeutet wie: setze mir das Sprite an eine bestimmte Position. Letztendlich können wir mit einer Interruptsteuerung überprüfen, ob zwei Sprites auf dem Bildschirm zusammengestoßen sind (ON SPRITE GOSUB).

Sprites lassen sich außer in SCREEN 0 auf allen anderen Bildschirmen ohne Probleme darstellen. Sie müssen allerdings darauf achtgeben, welche Spritegröße Sie mit dem zweiten Parameter im SCREEN-Befehl ansprechen wollen. Sind es die Stufen 0 und 1 (SCREEN ,0 bzw. SCREEN ,1), so haben wir die Möglichkeit, bis zu 256 Sprites in der Größe eines 8*8-Feldes (Buchstabengröße) zu definieren. Arbeiten wir jedoch in Stufe 2 oder 3 (SCREEN ,2 bzw. SCREEN ,3) können wir 'nur' bis zu 64 Sprites in der Größe 16*16 erstellen.

Schauen wir uns das Format des PUT SPRITE-Befehls einmal etwas genauer an:

```
PUT SPRITE A,(B,C),D,E
```

zu A: Hiermit wird eine Auswahl zwischen 0 und 31 getroffen, um dem MSX-Computer zu verstehen zu geben, in welcher Ebene wir arbeiten wollen. Sie müssen sich die 32 verschiedenen Sprites als durchsichtige Bildseiten vorstellen, die auf den Bildschirm (mit Grafik und Text) gelegt nur die definierten Spritemuster an der gewünschten Koordinate anzeigen. Sprite 31 liegt direkt auf unserem erstellten Bild, während Sprite 30 schon eine Stufe höher liegt, d.h. bei zwei Sprites, die sich bewegen und schließlich überlappen liegt Sprite 30 oben auf Sprite 31. Dies gilt entsprechend für die Hierarchie aller möglichen Sprites 0 bis 31. Die jeweilig niedrigere Spritenummer hat entsprechend höhere Priorität und liegt somit immer auf höherzahligeren Sprites. Pro Bildschirmebene bzw. um bei unserer Formulierung zu bleiben: pro durchsichtige Bildseite kann allerdings in jedem Fall nur 1 Sprite abgebildet werden d.h. wir müssen sämtliche Spritebildschirme von 0 bis 31 gleichzeitig zur Abbildung bringen, wollen wir die Spritedarstellunge ausreizen.

siehe Programm 4 im Anhang

zu B und C: Mit B wird die X-Koordinate, mit C die Y-Koordinate des darzustellenden Sprites angesprochen. Hierbei sind wir nicht darauf angewiesen, nur Werte im darstellbaren Bereich zu verwenden. Wir können vielmehr Werte für A und B zwischen -32768 und +32767 verwenden (noch größere bzw. kleinere Werte erzeugen einen 'Overflow error'); in bestimmten Zahlraumabständen erscheint das angesprochene Sprite dann immer wieder auf dem Bildschirm. Einerseits können wir die absolute Adressierung verwenden, indem wir z.B. zur Ansteuerung des Bildschirmmittelpunkts für A=128 und B=96 setzen:

```
PUT SPRITE 1,(128,96)
```

Es ist aber auch möglich (wie bei allen Grafikbefehlen), nicht absolut sondern relativ vorzugehen: Zu diesem Zweck sehen wir den augenblicklichen Grafikcursor als Nullpunkt an und sagen unserem MSX-Computer darüber Bescheid, wie viele Punkte er von dort nach rechts bzw. links und nach unten bzw. oben gehen soll. Um dem Computer darüber Bescheid zu geben, daß wir die relative der absoluten Adressierung vorziehen, müssen wir in den PUT

SPRITE-Befehl das Wörtchen STEP (=Schritt) integrieren z.B. um vom aktuellen Cursor 10 Punkte nach rechts und 10 Punkte nach oben zu gehen, müßte die PUT SPRITE-Anweisung lauten:

```
PUT SPRITE A,STEP(10,-10),D,E
```

Übrigens: Die angegebenen Koordinaten bei absoluter und relativer Adressierung beziehen sich in jedem Fall (ob kleine oder große Sprites) auf die äußerste linke obere Ecke des Sprite-Patterns (nicht erst auf den ersten sichtbaren Punkt des von Ihnen erstellten Spritemusters!).

zu D: Jedes Sprite darf nur einfarbig sein. Mit Parameter D wählen Sie für das Sprite die gewünschte Farbnummer aus, wobei Sie sämtliche zur Verfügung stehenden Farben von 0 bis 15 verwenden dürfen.

Mehrfarbige Sprites lassen sich nur dadurch erstellen, indem Sie verschiedene Sprites mit unterschiedichen Farben übereinanderlegen z.B. ein rotes Männchen (Sprite 0) mit grüner Jacke (Sprite 1) und blauer Hose (Sprite 2) wäre auf diese Art und Weise möglich:

siehe Programme FRETY und MISTER MINER im Anhang

Allerdings müssen Sie bei mehrfarbigen Sprites vorsichtig sein, denn:

a) der ON SPRITE-Befehl ist nicht mehr anwendbar, da sich bei mehrfarbigen Sprites andauernd die Sprites überlagern und somit ununterbrochen Meldungen über Spritekollisionen an den VDP in Ihrem MSX-Computer abgegeben würden.

b) unter BASIC ist die Spritesteuerung für mehrere übereinander liegende farbige Sprites manchmal so langsam, daß Sie ab und zu auf dem Bildschirm erkennen können, daß sich beispielsweise in unserem Männchen erst die rote Figur, dann die Jacke und schließ- und letztendlich die Hose verschiebt - ein Effekt, der sich nicht sehr leicht aus dem Weg räumen läßt.

c) Sie müssen Vorsicht walten lassen, wenn Sie ein Männchen aus drei oder mehr als drei verschiedenen Sprites zusammensetzen, denn Ihr MSX-Computer vrträgt nur vier Sprites pro Zeile. Kommt ein fünftes Sprite hinzu, wird das Sprite mit der höchsten Wertstufe aus- oder übergeblendet. Dies kann zu sehr unschönen Effekten führen.

siehe Programm 5 im Anhang

Sie fragen sich, warum bei 3-Sprite-Männchen diese Gefahr bereits besteht? Denken Sie daran, daß sich manchmal zwei Männchen in einem Spiel auf einer Bildschirmzeile begegnen ...

Dennoch sind mehrfarbige Sprites auch in Spielen möglich. Beide Spielprogramme, die in diesem Buch abgedruckt sind, arbeiten so. Vielleicht können Sie beim Abtippen noch dazulernen.

siehe Programme FRETY und MISTER MINER im Anhang

zu E: Mit dem zweiten Parameter des SCREEN-Befehls geben Sie Ihrem MSX-Computer darüber Bescheid, wie groß die Sprites auf dem Bildschirm dargestellt werden sollen: Sie können entweder 8*8-Sprites (SCREEN ,0 bzw. SCREEN ,1) oder 16*16-Sprites (SCREEN ,2 bzw. SCREEN ,3) definieren und auf den Bildschirm ausgeben.

Da im Computer nur ein begrenzter Speicherraum für Sprites zur Verfügung steht, können Sie entweder bis zu 256 verschiedene 8*8-Sprites oder aber 'nur' bis zu 64 verschiedene 16*16-Sprites definieren. Von diesen Sprites können Sie 32 gleichzeitig mit PUT SPRITE auf dem Bildschirm erscheinen lassen.

Mit Parameter E des PUT SPRITE-Befehls geben Sie an, welches vorher mit SPRITE$ definierte Sprite auf dem Bildschirm erscheinen soll. Dabei steht es Ihnen frei, ob alle 32 Sprites auf das gleiche Aussehen oder auf 32 verschiedene Aussehen im Speicher zurückgreifen sollen. Probieren Sie hierzu folgendes kurzes Programm einmal aus:

siehe Kurzlisting PUTSPRITE 1 im Anhang

Vorsicht gilt allerdings auch hier: wenn Sie zu 5 Spritedarstellungen in einer Horizontalen kommen, wird das höchstwertigste Sprite teilweise oder vollständig ausgeblendet (vertikal gibt es keine Grenze!).

siehe Programm 5 im Anhang

Noch ein paar Hinweise: Sie können die Kontrolle über mehr als 4 Sprites horizontal auch über die Abfrage von VDP(7) erreichen. Der Wert, den ein Bit von VDP(7) ausgibt (siehe unter VDP), bezieht sich dann auf die Nummer des höchstwertigsten Sprites, das ausgeblendet dann teilweise oder vollständig (nicht mehr) auf dem Bildschirm erscheint.

Was das Spriteaussehen beim PUT SPRITE-Befehl anbetrifft: Sie können die dazwischenliegende Farbangabe auch durch Kommaeingabe übergehen z.B.

```
PUT SPRITE 1,(128,96),,1
```

d.h. nun wird Spriteaussehen 1 aus dem Speicher zur Anzeige gebracht. In diesem Fall gilt die aktuelle Vordergrundfarbe auch als Spritefarbe.

In jedem Fall werden die zuletzt eingegebenen Werte verwendet, wenn Sie diese nicht eingeben; das betrifft sowohl die Farbe als auch die Spriteaussehensnummer.

Sind Ihnen die aktuellen Koordinaten Ihres Sprites bekannt (B,C), können Sie auf folgende Art und Weise auch nur die Farbe des Sprites ändern, ohne es auf dem Bildschirm zu verrücken:

```
DRAW "BM =B;,=C;"
PUT SPRITE 1,STEP(0,0),Farbe
```

Durch den BASE-Befehl erfahren Sie, wo zwei Speicherbereiche im VRAM beginnen, die für die Sprites lebensnotwendig sind:

1. Der Sprite Attribute Table, der Informationen über Spriteaussehensnummer, aktuelle Koordinaten und Spritefarbe enthält (das sind die Daten, die wir mit PUT SPRITE über BASIC ansprechen).

2. Der Sprite Pattern Table, der Informationen über das Spriteaussehen enthält (die Daten, die wir mit SPRITE$ über BASIC erzeugen).

Wollen Sie nicht die BASIC-Befehle PUTSPRITE und SPRITE$ verwenden, können Sie die gewünschten Werte auch in die entsprechenden Adressen der Tables mit VPOKE eingeben bzw. mit PRINT VPEEK diese Informationen auch aus dem Speicher einlesen (z.B. welche Koordinate und welche Farbe hat z.Z. Sprite 3?). Hierzu müssen Sie über den Aufbau des VRAM genauestens Bescheid wissen; schauen Sie dazu bitte vorher unter BASE, SCREEN und VDP in diesem Buch nach.

Sie können jedes Sprite kurzfristig vom Bildschirm verschwinden lassen, indem Sie für die Y-Koordinate den Wert 209 eingeben:

```
PUT SPRITE 1,(X,209)
```

Durch eine gültige Y-Koordinateneingabe erscheint dasjenige Sprite dann wieder an der entsprechenden Stelle auf dem Bildschirm.

Die können die Sprites mit höherer Spritenummer kurzfristig vom Bildschirm verschwinden lassen, wenn Sie in die Y-Koordinate den Wert 208 eingeben:

```
PUT SPRITE 1,(X,208)
```

d.h. alle Sprites mit einer größeren Spritenummer als 1 verschwinden augenblicklich vom Bildschirm.

Sie können diesen Zustand wieder beseitigen, indem Sie in unserem Fall für Sprite 1 eine gültige Y-Koordinate eingeben.

PUTSPRITE 1

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 1,0
30 VDP(6)=VDP(4)
40 PRINT "Zuerst 5 Sprites mit"
50 PRINT "unterschiedlichem Muster:
60 FOR N=0 TO 191
70 PUT SPRITE 1,(128,N+8)
80 PUT SPRITE 2,(128,N+16)
90 PUT SPRITE 3,(128,N+24)
100 PUT SPRITE 4,(128,N+32)
110 PUT SPRITE 5,(128,N+40)
120 NEXT N
130 CLS
140 PRINT "Nun 5 Sprites mit"
150 PRINT "gleichem Muster:"
160 FOR N=0 TO 191
170 PUT SPRITE 1,(128,N+8),,1
180 PUT SPRITE 2,(128,N+16),,1
190 PUT SPRITE 3,(128,N+24),,1
200 PUT SPRITE 4,(128,N+32),,1
210 PUT SPRITE 5,(128,N+40),,1
220 NEXT N
230 RUN
```

Befehle Sound PLAY
Steuerungsbefehl für die MSX-Musikprogrammiersprache

siehe hierzu auch die Programme im Anhang: Programm
Programm 9
Programm 10
PLAY(N) 4

Neben einer Vielzahl von Sound- und vor allen Dingen Grafikbefehlen bietet das MSX-BASIC auch zwei eigene programmierbare Sprachen, die uns die Ansteuerung jedbeliebiger Effekte mit Sound- und Grafikprozessor sehr erleichtern:

Für Grafik die GML (Graphic Macro Language)
Für Sound die MML (Music Macro Language)

Während die Grafiksprache mit dem Befehl DRAW arbeitet, den wir an anderer Stelle in diesem Buch ausführlich erläutert haben, bedient sich die hier besprochene Musiksprache des Befehls PLAY, was soviel bedeutet wie: 'Spiele Noten'.

Sowohl die Grafik- als auch die Musiksprache arbeiten mit Strings, was bedeutet, daß die nach dem Befehl PLAY bzw. DRAW folgenden Befehlszeichen in Anführungsstriche eingeschlossen werden müssen. Dies jedoch hat den großen Vorteil, daß wir die einmal definierten Klänge verschiedenen Variablen zuweisen können und diese nach eigenem Belieben mit Stringbefehlen (MID$, LEFT$, INSTR, RIGHT$ ...) umwandeln und auf bis zu 255 Zeichen pro String erweitern können.

Sämtliche Teilbefehle, die in dem dem Befehl PLAY nachfolgenden String enthalten sein können, müssen nur im Ausnahmefall durch Semikolons bzw. Kommata voneinander abgetrennt werden. Wenn diese Abtrennung erfolgen muß, verweisen wir selbstverständlich in diesem Kapitel auf die zwingende Notwendigkeit. Ist dies aber nicht ausdrücklich vorgeschrieben, können Sie alle Zeichen im PLAY-String direkt ohne Leerzeichen hintereinander schreiben. Der Übersichtlichkeit halber haben wir aber in diesem Kapitel zusätzliche Leerzeichen eingefügt, jedoch können Sie auch alles ohne Leerzeichen hintereinander schreiben und sparen dadurch

natürlich im Laufe von größeren Programmen viel Speicherplatz.

So wie wir das bereits in der Schule gelernt haben, so können wir unserem MSX-Computer auch Töne entlocken. Geben Sie

PLAY "CDE"

ein und es erklingen die Noten C,D und E. Bis zu 255 Noten können Sie auf diese Art nacheinander in einen String schreiben.

Sollen gar zwei Tonreihen parallel zueinander erklingen, müssen Sie nur die Tonreihen wie zwei einzelne Strings voneinander durch ein Komma trennen:

PLAY "CDE","DEF"

und es erklingen parallel zueinander die Tonpaare CD, DE und EF. Auch hier wäre es theoretisch möglich, daß Sie 255 Töne nacheinander parallel zueinander durch einen PLAY-Befehl erklingen lassen.

Die Höhe der Gefühle bei den MSX-Computern heißt letztendlich Dreiklang. Der dritte Tonkanal wird hierbei wiederum von den zwei anderen Tonkanälen durch ein weiteres Komma getrennt:

PLAY "C","E","G"

Nicht mehr der Rede wert ist an dieser Stelle erneut der Hinweis: bis zu 255 Noten lassen sich parallel zueinander durch jeweils einen PLAY-Befehl pro Musikkanal abspielen!

Zur Verfügung stehen uns beim Gebrauch des PLAY-Kommandos die allgemein gebrächlichen Tonbezeichnungen:

C D E F G A B

Was bei einem Klavier die schwarzen Tasten sind (jeweils ein halber Ton höher bzw. tiefer als die benachbarten weißen Tasten) sind in der Music Macro Language die Plus- bzw. Minuszeichen.

Eine vollständige Tonleiter ohne Halbtöne würde so aussehen:

PLAY "CDEFGAB"

und im Vergleich dazu eine Tonleiter mit sämtlichen Halbtönen:

PLAY "CC+DD+EE+FF+GG+AA+BB+"

Ihnen wird aufgefallen sein, daß hier etwas nicht stimmt und da sehen Sie ganz richtig:

1) Der Ton "E+" entspricht dem Ton F
2) Der Ton "B+" entspricht dem Ton C

Warum dies so ist, zeigt uns ganz einsichtig das Abbild einer Klaviertastaturoktave:

siehe Programm 9 im Anhang

Genau sieben Ganztöne und fünf Halbtöne lassen sich in einer Oktave unterbringen und ein erhöhtes "E" gibt es halt nicht, anstatt dessen ist die nächsthöhere und damit gleichwertige Note das "F".

Wollen wir über das "B" hinaus einen Halbton höher spielen, gibt es wiederum keine durch eine entsprechend schwarze Taste zu erzeugenden Ton; der nächste mögliche Ton ist halt wieder das "C", allerdings um eine Oktave höher als die, in der wir gerade musizieren.

Der Vollständigkeit halber hier unsere richtige Oktave und noch eine andere Möglicheit, wie wir eine Oktave samt allen möglichen Halbtönen erklingen lassen können:

1) PLAY "CC+DD+EFF+GG+AA+B"
2) PLAY "CD-DE-EFG-GA-AB-B"

Wie in der musikalischen Notation üblich, können wir an Stelle des Pluszeichens (für eine Halbtonerhöhung) auch ein Kreuzchen setzen, so daß unsere Tonfolge von vorhin auch so aussehen kann:

1) PLAY "CC#DD#EFF#GG#AA#B"

Wollen wir einen Halbtonschritt anzeigen, muß das entsprechende Plus-, Minus- oder Kreuzchenzeichen, wie Sie den aufgeführten Beispielen entnehmen können, immer hinter dem Notennamen stehen, andernfalls würde sich das Zeichen auf die vorherige Note (falls vorhanden) beziehen.

Unserem einen Fehler von vorhin sind wir nun ja schon mit Hilfe des Abbildes einer Klaviertastatur in Programm 8 auf die Schliche gekommen: Nicht alle möglichen Halbtöne ergeben zu den bereits vorhandenen Tönen eine neue Tonhöhe. Wie steht es denn mit der Tatsache, daß wir zu Anfang in unserer Tonfolge für das erhöhte "B" wieder ein tiefes "C" erhielten?

Wir müssen unserem MSX-Computer also auch Bescheid darüber geben, in welcher Oktave der Ton jeweils zu erklingen hat; in unserem Beispiel vorhin war dazu keine Angabe erfolgt, so daß der Computer seinen Standardwert übernommen hat und d.h. soviel wie: Spiele in Oktave 4. In Wirklichkeit können wir aber über gleich 8 Oktaven verfügen, die wir im PLAY-Befehl durch das Einfügen von "O" und einer Zahl zwischen 1 und 8 deutlich machen. Probieren wir also einfach mit diesem neu dazugelernten PLAY-Kommando, das erhöhte "B" ("B+") auch wahrlich erhöht d.h. in einer Oktave höher als "C" wiederzugeben:

PLAY "CC+DD+EFF+GG+AA+B O5 B+"

Wunderschön, es klappt ... und es klappt noch einmal gespielt nicht mehr! Gehen Sie mit dem Cursor auf dem Bildschirm hoch und lassen die Melodie ein weiteres Mal erklingen: Diesmal wird die gesamte Tonfolge in Oktave 5 gespielt, was heißt, daß unsere letzte Note wieder zum tiefen "C" zurückkehrt. Der MSX-Computer hat sich sofort nach Lesen des "O5"-Befehls eingeprägt, daß von nun an alle Töne in Oktave 5 zu erklingen haben. Wir könnten nun einerseits Oktave 5 durch Oktave 6 ("O5" wird zu "O6") ersetzen,

aber das wäre nicht unbedingt im Sinne des Erfinders, weil es von nun an (bis Oktave 8) immer höher hinaus geht.

Besser gelingt es uns, wenn wir vor die zuerst gespielten Töne die Oktavennummer 4 einfügen, während vor dem zuletzt gespielten Ton die Angabe "O5" (eine Oktave höher als "O4") beibehalten wird:

```
PLAY "O4 CC+DD+EFF+GG+AA+B O5 B+"
```

Nun ist es richtig und Sie können nach Belieben die Oktaven wechseln (von Oktave 1, die mehr oder minder brummig klingt, bis hinauf zu Oktave 8 in den höchsten Tönen). Es sei Ihnen anheimgestellt, ob Sie nun vor jedem Notennamen die Oktave durch Einfügen einer entsprechenden "O"-Zahl verändern, oder aber nur einmal vor der dazu bestimmten Notenfolge. Ihr MSX-Computer akzeptiert selbstverständlich beide Möglichkeiten.

Wir hatten zu Beginn dreistimmige Kompositionen (Dreiklang) ertönen lassen. Der eingegebene Oktavenwert gilt in jedem Fall, wie auch die im weiteren Verlauf dieses Kapitels angegebenen Tonveränderungen, immer nur für den einen gerade angewählten Tonkanal. Soll die Änderung in sämtlichen Tonkanälen stattfinden, müssen wir sie auch entsprechend dort einzeln anbringen: Somit würden also folgende 3 Tonkanäle den selben Dreiklang in verschiedenen Tonhöhen erzeugen:

```
PLAY "O4 C O5 C O6 C","O4 E O5 E O6 E","O4 G O5 G O6 G"
```

Merken Sie sich aber in jedem Fall: Wenn Sie ohne Oktavenänderungseingaben in einer Oktave bleiben wollen, springen Sie über das "B" (denn "B+" ist das tiefe "C") und das "C" ("C-" ist das hohe "B") nie hinaus.

Für die Notennamen-Unkundigen ist es auch möglich, daß Sie in der Music Macro Language entsprechend der Tonhöhe mit Zahlen arbeiten können: Die Zahlen werden in dem PLAY-Befehl durch ein vorangestelltes "N" (für Number = Nummer) eingegeben. Sie können sämtliche Noten aller zur Verfügung stehenden Oktaven also auch mit Zahlen anspringen, d.h., das "C" der ersten Oktave ist

Nummer 1 ("N1") und das "B" der achten = höchsten Oktave Nummer 96 ("N96"). Da jede Oktave aus 12 Tönen besteht ("CC+DD+EFF+GG+AA+B") und wir über 8 Oktaven verfügen könnnen, gibt es nachweislich - wie auch schon angegeben - 8*12=96 Töne. Zur Orientierung: Unser "C" in Oktave 4 ist "N36" und die gesamte Oktave würde mit Zahlangabe so aussehen:

```
PLAY "N36 N37 N38 N39 N40 N41 N42 N43 N44 N45 N46 N47"
```

Sie werden sich nun sicherlich fragen, warum man zu der übersichtlichen Eingabe der Notennamen auch noch diese recht unübersichtliche Zahleingabe benötigt. Dies hat mehrere Gründe:

1) Wir brauchen uns um keine Oktaveneingabe zu kümmern. Die Eingabe erfolgt absolut mit Zahlwerten, was soviel heißt wie: Je höher die Zahl, desto höher auch der Ton.

2) Wenn wir im weiteren Verlauf dieses Kapitels kennenleren, wie ein String vom Computer bearbeitet werden kann (Aufteilung in bzw. Zusammensetzen durch PLAY-Teilstrings), ist es allemal einfacher, mit Zahlwerten zu arbeiten als mit Notennamen, die dem Computer in ihrer Reihenfolge nicht so leicht zu erklären sind (die Folge "CDEFG" ist ja noch eindeutig und errechenbar, da sie dem Alphabet folgt, wie sieht es aber mit den nachfolgenden Notennamen "A" und "B" aus?). Hier haben wir Menschen diese willkürliche Folge auswendig gelernt bis sie uns in Fleisch und Blut überging. Einem Computer (der wohlgemerkt immer nur so schlau ist, wie die Leute, die ihn programmiert haben) ist gar nicht so leicht zu erklären, daß auf eine G wieder ein A folgt.

Bisher spielte unser MSX-Computer die Noten mit jeweils gleich langen Tönen wie ein Kind, das seine ersten Geigentöne beigebracht bekommt. Variieren läßt sich aber ein Ton nicht allein in der Tonhöhe, sondern auch in der Tonlänge: Jeder Ton erklingt nach dem Einschalten des Computers als eine Viertelnote, was mit dem nun zu besprechenden PLAY-Befehl "L" heißen würde:

```
PLAY "L4"
```

Während "L4" für eine Viertelnote steht, können wir auch "L8" vor einen Notennamen setzen, was soviel bedeutet wie: die folgenden Noten (bis zur nächsten Tonlängenänderung durch "L") sind Achtelnoten. "L1" würde bedeuten: Ganze Noten, "L2" halbe Noten ... bis hin zu "L64" = Vierundsechzigstel Noten (eine ganze Note besteht in diesem Fall aus 64 Noten mit der Tonlänge 1/64).

In unserem folgenden Beispiel erklingen sämtliche drei Tonkanäle geichlang, aber alle Töne haben halt eine andere Tonlänge:

Kanal 1 = 8mal 1/8 Note
Kanal 2 = 4mal 1/4 Note
Kanal 3 = 1mal 1/1 Note

Um den Unterschied deutlich und damit hörbar zu machen, weisen wir jedem Kanal eine andere Oktave zu:

```
PLAY "O4 L8 CDECDECD","O5 L4 CDEC","O6 L1 C"
```

Ähnlich wie beim Oktavenbefehl, merkt sich unser MSX-Computer auch die Tonlänge bis zur nächsten Änderung des Tonlängenwerts. Sollen Tonlängenänderungen aber nur für einzelne Töne gelten, lassen wir die "L"-Angabe weg und geben anstatt dessen hinter dem Notennamen die gewünschte Tonlänge ein. Also hätten folgende Töne die gleiche Tonhöhe und -länge:

```
PLAY "C32"
PLAY "L32 C"
```

Im zweiten Beispiel wird durch "L32" vorgegeben, daß auch alle zukünftigen Töne (bis zur nächsten "L"-Eingabe) ebenfalls als 32stel Noten zu erklingen haben.

Es ist auch möglich, den Buchstaben für Länge "L" bei einzelnen Notenlängenänderungen wegzulassen. In diesem Fall müssen wir allerdings zur Abtrennung zwischen den Zahlwerten (einerseits für Tonhöhe, andererseits für Tonlänge) ein Semikolon setzen:

Eingabe von Notennamen: PLAY "C8 DE"
Eingabe von Notennummern: PLAY "N36;8 N38 N40"

Wir können die Länge einer Note auch noch auf eine andere Weise verlängern: Wie in der Musikschreibweise besagt ein Punkt, der einer Note folgt, daß dieselbige um die Hälfte länger zu spielen ist: So werden aus einer ganzen Notenlänge eineinhalb Notenlängen, aus einer halben Notenlänge eine dreiviertel usw:

PLAY "L4 C." = eine Dreiviertelnte
PLAY "L1 C." = eine ganze und eine halbe Note

Zum Musikmachen brauchen wir auch die Möglichkeit, keine Musik zu machen. Dieses Unterbrechen (Pause, im Englischen heißt Pause "Rest") werden in unseren Tonfolgen durch den Buchstaben "R" (für Englisch = "Rest") gekennzeichnet. Dem "R" folgt wiederum eine Zahlangabe zwischen 1 (für eine ganze Pause) und 64 (für eine vierundsechzigstel Pause):

PLAY "C R4 DE"

Ton "C" erklingt, eine Viertel Pause lang ist Stillschweigen und schließlich hören wir die Töne "D" und "E".

Besonders sinnvoll ist die Eingabe von extrem kurzen Pausenwerten, wenn mehrmals der gleiche Ton in Folge erklingt: Während sich im folgenden Beispiel 4 Töne wie ein langer Ton anhören:

PLAY "CCCC"

bewirkt die kurze Pause ("R64"), daß wir sämtliche 4 Töne in ihrem Anschlag voneinander unterscheiden können:

PLAY "C R64 C R64 C R64 C"

Würden Sie dieses Vorgehen jedoch bei der Eingabe mehrstimmiger Musik heranziehen, hätten Sie alsbald unter der Asynchron-Wiedergabe der drei Tonkanäle zu leiden.

Wir können auch den Buchstaben "R" ohne Zahlangabe in den String einbauen. So nimmt der Buchstabe "R" den ihm zugeordneten Standardwert ein, was soviel bedeutet wie: Die Pause dauert eine viertel Notenlänge.
Bevor wir zur direkten Änderung des Tones (Hüllkurve usw.) kommen, müssen wir uns noch mit dem Tempo und der Lautstärke und ihrer Bedeutung auseinandersetzen: Bisher haben wir Viertel-, Halb- oder gar Vierundsechzigstel-Noten kennengelernt und dabei so getan, als wenn z.B. eine Viertelnote in jedem Musikstück so und so viele Bruchteile einer Sekunde zu erklingen habe. Dies stimmt aber so nicht, denn das vorgegebene Tempo allein (rhythmische Bewegung des Dirigenten mit seinem Taktstock) bestimmt auch die Schnelligkeit des gespielten Musikstückes und seiner Noten.

Befindet man sich allerdings in einem vorgegebenen Tempo, haben die Noten relativ zueinander wieder die gleiche Länge:

eine halbe Note ist genauso lang wie zwei Viertelnoten
eine ganze Note ist so lang wie acht Achtelnoten usw.

Unser MSX-Computer hat beim Einschalten den Wert 120 für das Tempo vorgegeben. Diesen Wert können wir jederzeit zurückrufen durch Eingabe des Buchstabens "T" ohne eine darauf folgende Zahleingabe. Andernfalls ist es möglich, das Tempo unseres Musikstückes zu erhöhen (bis "T255") oder halt zu verlangsamen (bis "T32"). Schauen bzw. hören wir uns dies an folgenden Beispielen einmal an:

```
PLAY "T CDE"        Tempo = 120 = normal
PLAY "T240 CDE"     Tempo = 240 = schnell
PLAY "T60 CDE"      Tempo = 60 = langsam
```

Die Lautstärke unserer Musik läßt sich durch Eingabe des Buchstabens "V" (für Englisch "Volume" = Lautstärke) und einer darauf folgenden Zahl zwischen 0 und 15 eingeben. Geben wir

keinen "V"-Befehl nach dem Einschalten des Computers ein oder setzen lediglich "V" ohne Zahlangabe in einen Musikstring, erklingen die darauf folgenden Töne bis zur nächsten Lautstärkeänderung mit "V"=8 bzw. Lautstärke=8.

Lauter werden die Töne bei Einfügung von "V" bis 15, leiser bei Einfügung von "V" bis 1 bzw. "V" bis 0, was bedeuten würde: kein Ton erklingt:

```
PLAY "V CDE"        Lautstärke = 8 = normal
PLAY "V15 CDE"      Lautstärke = 15 = laut
PLAY "V1 CDE"       Lautstärke = 1 = leise
PLAY "V0 CDE"       Lautstärke = 0 = aus
```

Wie im Kapitel zum Befehl SOUND erklärt wird, können wir jeden Ton mit Hilfe von verschiedenen Hüllkurven in seinem Klang verändern. Zum Beispiel ist es möglich, daß ein Ton nicht sofort erklingt ist, sondern langsam lauter wird (in seiner Lautstärke bis zum Höchstpunkt ansteigt). Eine andere Möglichkeit wäre die, einen Ton kurz nacheinander auf- und abschwingen zu lassen.

Um diese Tonänderungen durchzuführen, haben wir mit dem Buchstaben "S" (für Englisch "Shape") acht verschiedene dieser sogenannten Hüllkurven zur freien Verfügung. Um eine Hüllkurve erklingen zu lassen, ist es sinnvoll, die Notenlänge auf 1 (L1=ganze Note) einzustellen, die V(olume=Lautstärke)-Eingabe vollkommen zu übergehen (Ihr MSX-Computer arbeitet nämlich bei Hüllkurvenanwahl stetig mit V16) und natürlich noch ein paar Töne hinzuzufügen (damit überhaupt etwas zu hören ist, denn die Hüllkurve erklingt nicht ohne Ton sondern nur mit bzw. durch den Ton). Folgende Eingabe wäre z.B. denkbar:

```
PLAY "S10 L1 CDEFG"
```

Hören Sie sich folgende Hüllkurven einmal genauer an:

S1, S4, S8, S10, S11, S12, S13, S14

Die Töne "C", "D" und "E" durch eine Hüllkurve zu einem Hubschrauberschnattergeräusch verändert würde beispielsweise so aussehen:

PLAY "S8 CDE"

oder das Anschlagen von Trommeln:

PLAY "S1 CDE"

Sie können zudem jede Hüllkurve noch in ihren Feinheiten verändern, indem Sie eine besondere Abstimmung mit Hilfe des Buchstabens "M" und einer darauf folgenden Zahl zwischen 1 und 65535 vornehmen.

So bekommen unsere "Trommeln" vom vorherigen Beispiel ein wenig mehr Volumen und Klang, wenn wir sie folgendermaßen ansprechen:

PLAY "S1 M2000 CDE"

Warum dies so ist, werden wir noch unter dem SOUND-Befehl näher erläutern. Hier soll erst einmal der Hinweis darauf ausreichen.

Damit haben wir alle Möglichkeiten aufgeführt, um den PLAY-Befehl, oder sagen wir lieber "die PLAY-Programmiersprache" sinnvoll zu nutzen. Der Übersichtlichkeit halber haben wir hier noch einmal alle Parameter aufgeührt:

C D E F G A B	Tonname
+ - #	Tonhöhenveränderung um einen halben Ton
O	Auswahl zwischen 8 Oktaven
N	Zahlanwahl für Tonhöhe (1 bis 96)
L	Tonlänge (1 = 1/1 bis 64 = 1/64)
.	Tonlänge um die Hälfte verlängern
R	Pause (1 = 1/1 bis 64 = 1/64)
T	Tempo (32 bis 255)
V	Lautstärke (0 bis 15)
S	Hüllkurve (1,4,8,10,11,12,13,14)
M	Abstimmung der Hüllkurven (1 bis 65535)

Es ist Ihnen sicherlich schon aufgefallen, daß Ihr MSX-Computer eine Tonfolge (auch gleichzeitig in drei Stimmen) spielen kann, während der Cursor bereits zur weiteren Eingabe bereit ist. Dies hängt ganz einfach damit zusammen, daß Ihr MSX-Computer einen extra Speicher besitzt, der sich eine Tonfolge von bis zu 24 Tönen (dreistimmig) merken und suksessive abspielen kann, ohne den weiteren Programmablauf zu stören. Wird allerdings dieses Merklimit überschritten, kann es lange dauern, bis Ihr Programm in normaler Geschwindigkeit weiterläuft (siehe auch unter PLAY(N) nach).

siehe Kurzlisting PLAY(N) 4 im Anhang

Außerdem wird Ihnen, wie bereits zu Beginn dieses Kapitels kurz darauf hingewiesen, aufgefallen sein, daß wir es bei den PLAY-Befehlen mit normalen Strings zu tun haben. So ist es möglich, wie im folgenden Programm gezeigt, vorgegebene Strings einfach der Reihe nach zu spielen:

```
10 A$="CDE"
20 B$="FGA"
30 PLAY A$,B$
```

Wir können diese Strings aber auch in bestehende PLAY-Anweisungen direkt einbinden:

```
PLAY "CDE X B$;"
```

Hierbei muß der Unterstring durch ein "X" eingeleitet und durch ein ";" (Semikolon) abgeschlossen werden.

Wir können auch direkt Variablen in unseren PLAY-Befehl vom Programm her einlesen (ähnlich wie bei der Einbindung des Unterstrings geben Sie vor der Variablen ein "=" ein und schließen den Variablennamen mit einem ";" ab), z.B. erklingen anschließend sämtliche 96 Töne von "N1" bis "N96":

```
10 FOR N=1 TO 96
20 PLAY "N =N;"
30 NEXT N
```

Schneller läuft das Programm, wenn wir sowohl das Tempo als auch die Tonlänge auf die schnellstmöglichen Werte festlegen:

```
20 PLAY "T255 L64 N =N;"
```

In Zeile 20 wurde die übernommene Variable "N" (aus der FOR ... NEXT - Schleife von Zeile 10) durch ein Gleichheitszeichen und ein Semikolon, beides zur Abtrennung vom übrigen String, umgeben.

Noch ein Hinweis: Durch die Tastenkombination µCTRL° µSTOP° zum Abbruch eines Programms sowie durch das BEEP-Kommando werden alle PLAY-Parameter (Oktave, Lautstärke ...) wieder auf ihren Urzustand (Oktave = 4, Lautstärke = 8 ...) zurückgesetzt. Das ist manchmal dazu ganz gut zu wissen, wenn Sie nicht mehr wissen, wie Sie aus Ihrer PLAY-SOUND-Abänderung herauskommen sollen.

Außerdem sei darauf hingewiesen, daß sowohl der SOUND- als auch der BEEP- und natürlich der PLAY-Befehl auf die gleichen Register des Soundprozessors zurückgreifen. Bis auf wenige Ausnahmen bedingt also die Eingabe eines dieser Befehle auch die Ausgabeänderung bei einem anderen (z.B. BEEP setzt alle Sound-Register auf den Anfangswert zurück).

Befehle Grafik PLAY (N)
Überprüfungsbefehl, ob Töne gespielt werden oder nicht

siehe hierzu auch die Programme im Anhang: PLAY(N) 1
PLAY(N) 2
PLAY(N) 3
PLAY(N) 4

Wie wir bereits bei der Behandlung der Programmiersprache mit PLAY angeführt haben, besitzen die MSX-Computer einen gesonderten Speicherbereich, in dem sie sich bis zu 25 Töne (dreistimmig) merken können. Nachdem diese Töne durch die PLAY-Programmiersprache dem MSX-Computer deutlich gemacht worden sind, läuft das BASIC-Programm weiter und - im Hintergrund spielt die eingelesene Melodie.

Dieser Vorgang läuft für uns wie selbstverständlich ab, jedoch sollten wir im Auge behalten, daß uns das MSX-BASIC mit dieser Fähigkeit, zwei Vorgänge gleichzeitig ablaufen zu lassen, etwas ganz exklusives an die Hand gibt: In Wirklichkeit handelt es sich bei diesem Vorgehen nämlich um eine sogenannte Interruptsteuerung, was soviel bedeutet wie: das BASIC wird andauernd durch eine im Hintergrund ablaufende Maschinensprachroutine unterbrochen, ohne daß wir uns im BASIC-Programm darum kümmern müssen.

Das MSX-BASIC zeichnet sich durch eine Vielzahl von solchen Interruptsteuerungen aus: einmal wird dadurch ständig eine mögliche Sprite-Kollision überprüft (ON SPRITE), ein anderes Mal das Drücken einer Funktionstaste (ON KEY) und schließlich die passende Reaktion des Programms bei einem plötzlich auftretendem Error (ON ERROR).

Nun aber wieder zurück zu unserem PLAY-Befehl: die 25-Ton und - Interruptsteuerung läßt sich überwachen d.h. wir können nicht nur mit unserem Gehör feststellen, wann die Melodie aufgehört hat zu spielen (d.h. die 25 im Hintergrund spielenden Töne zu Ende gehen), wir können dazu auch den Speicher befragen. Dazu dient uns der PLAY(N)-Befehl bzw. eigentlich müßte man sagen die PLAY(N)-Variable: spielen gerade Töne über irgendeinen der drei

zur Verfügung stehenden Musikkanäle, so ist die Aussage wahr = in Computerrechenweise = -1; ist dies nicht der Fall, so erfolgt als Antwort auf die Frage nach Ton oder Nicht-Ton falsch = 0. Probieren wir dies doch einfach einmal aus:

siehe Kurzlisting PLAY(N) 1 im Anhang

So lange die Musik ertönte, so lange wurde auch die Zahl -1 auf dem Bildschirm angezeigt (-1 = wahr = die Musik spielt). Hörte die Musik schließlich auf, wurde der Wert 0 auf dem Bildschirm angezeigt (0 = falsch = die Musik spielt nicht mehr).

Wie bei so vielen MSX-Befehlen bzw. -Variablenabfragen brauchen wir uns auch im Fall der PLAY(N)-Anweisung nicht damit zufrieden zu geben, daß nur eine allgemeine Feststellung getroffen wurde wie: Musik an - Musik aus.

Mit Hilfe des Parameters können wir folgende Unterscheidungen treffen:

PLAY(0): Abfrage aller 3 Kanäle, ob an/aus, -1/0
PLAY(1): Abfrage von Kanal 1, ob an/aus, -1/0
PLAY(2): Abfrage von Kanal 2, ob an/aus, -1/0
PLAY(3): Abfrage von Kanal 3, ob an/aus, -1/0

Wie leicht kann es denn bei einem Musikstück passieren, daß zwar Kanal 1 mit der Hauptmelodie fröhlich weiterdudelt, während Kanal 2 und Kanal 3 bereits wieder in Stellung sind, um neue Musikkommandos anzunehmen. Auch diese Prozedur soll im folgenden Programm auf dem Bildschirm angezeigt werden:

siehe Kurzlisting PLAY(N) 2 im Anhang

In Zeile 40 erklingt eine dreistimmige Melodie, in der zum Demonstrationszweck jeder Kanal eine unterschiedliche Anzahl von zu spielenden Tönen vorweist. Lassen wir das Programm mit RUN laufen, werden auf dem Bildschirm anfangs viermal die Worte "an" erscheinen; nach und nach hören die Kanäle mit dem Spielen auf, woraufhin auf dem Bildschirm auch eine entsprechede Reaktion zu sehen ist ("aus").

Vielleicht wundern Sie sich über die kurze IF ... THEN-Abfrage in den Zeilen 90 bis 120, deshalb hier ein Hinweis:

IF A THEN...

bedeutet: Wenn A wahr ist d.h. A=-1 (d.h. mindestens einer der drei Tonkanäle ist angestellt), dann schreibe auf den Bildschirm "an". Der MSX-Computer sieht laut Boolescher Algebra die geforderte Bedingung erfüllt, oder nicht erfüllt und reagiert darauf entsprechend.

Die hier besprochene PLAY-Abfrage ermöglicht es uns, bei entsprechend klugem Taktiern jeweils den Musikkanal so voll zu machen, daß das Programm weiter ablaufen kann, ohne daß die gespielte Musik für einen längeren Zeitaufenthalt sorgt.

Schauen wir uns einmal an, ob wahrlich alle drei Musikkanäle für 25 Töne lang ihre Arbeit eigenständig verrichten:

siehe Kurzlisting PLAY(N) 3 im Anhang

Zum Schluß noch ein kleiner Hinweis, der Ihnen als Herausforderung gelten soll:

Im RAM beginnt bei Speicherstelle 63830 der Bereich, in dem die Daten des PLAY-Befehls abgespeichert sind. Versuchen Sie einmal herauszufinden, was sich während Sie eine Melodie mit PLAY spielen lassen, dort alles ereignet.

Ein Tip zur Entschlüsselung von Teilen dieses Speicherbereichs sei schon hier einmal angebracht.

Sie können die Musikkanäle 1,2 und 3 in ihrem Spieldrang unterbrechen, wenn Sie folgendes tun:

1. Musikkanal 1 ausstellen:

POKE 63833,255 POKE 63834,255

2. Musikkanal 2 ausstellen:

POKE 63839,255 POKE 63840,255

3. Musikkanal 3 ausstellen:

POKE 63845,255 POKE 63846,255

Da diese Speicherstellen zu dem Zeitpunkt den Wert 0 tragen, wenn kein Ton erklingt, können Sie durch Abfrage dieser Speicherstellen unseren PLAY(N)-Befehl ersetzten. Sie setzen einfach eine Abfrageschleife in Gang und bekommen als Plus zum PLAY(N)-Befehl sogar noch genau mitgeteilt, wie viele Töne in jedem Kanal bis jetzt gespielt wurden und wie viele Töne insgesamt zu spielen beabsichtigt wird.

siehe Kurzlisting PLAY(N) 4 im Anhang

PLAY(N) 1

```
            -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN Ø
30 PLAY "cdefgab"
40 PRINT PLAY(Ø)
50 GOTO 40
```

PLAY(N) 2

```
            -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN Ø
30 WIDTH3Ø
40 PLAY "cdefgab","defgab","efgab"
50 A=PLAY(Ø)
60 B=PLAY(1)
70 C=PLAY(2)
80 D=PLAY(3)
90 IF A THEN PRINT "an  ";ELSE PRINT "au
s ";
100 IF B THEN PRINT "an  ";ELSE PRINT "a
us ";
110 IF C THEN PRINT "an  ";ELSE PRINT "a
us ";
120 IF D THEN PRINT "an  "ELSE PRINT "au
s "
130 GOTO 50
```

PLAY(N) 3

```
            -*-*-*-*-*-
10 CLEAR 2000
20 COLOR 1,15
30 SCREEN Ø
40 WIDTH37
50 FOR N=1 TO 25
60 A=INT(RND(1)*96)
70 B=INT(RND(1)*96)
80 B=INT(RND(1)*96)
90 A$=A$+"N"+STR$(A)
100 B$=B$+"N"+STR$(B)
110 C$=C$+"N"+STR$(C)
120 NEXT N
130 PLAY A$,B$,C$
140 PRINT "25 Toene in 3 Musikkanaelen s
pielen  von nun an eigenstaendig; Sie ko
ennenwaehrenddessen weiter programmieren
  (oder zaehlen Sie ruhig einmal mit)"
```

PLAY(N) 4

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 0
30 WIDTH 37
40 PLAY "cdedcdeccdecdecdecdecdec","cede
decdededededcdede","cdedededededededede"
50 LOCATE 0,0
60 PRINT INT(PEEK(63834!)/5)".Ton aus Ka
nal 1"
70 LOCATE 0,1
80 PRINT (PEEK(63833!)-1)/5"Toene erklin
gen aus Kanal 1"
90 PRINT
100 PRINT INT(PEEK(63840!)/5)".Ton aus K
anal 2"
110 PRINT (PEEK(63839!)-1)/5"Toene erkli
ngen aus Kanal 2"
120 PRINT
130 PRINT INT(PEEK(63846!)/5)".Ton aus K
anal 3"
140 PRINT (PEEK(63845!)-1)/5"Toene erkli
ngen aus Kanal 3"
150 PRINT
160 IF PEEK(63833!)=PEEK(63834!) THEN PR
INT "Kanal 1 ist aus" ELSE PRINT "Kanal
1 ist an"
170 IF PEEK(63839!)=PEEK(63840!) THEN PR
INT "Kanal 2 ist aus" ELSE PRINT "Kanal
2 ist an"
180 IF PEEK(63845!)=PEEK(63846!) THEN PR
INT "Kanal 3 ist aus" ELSE PRINT "Kanal
3 ist an"
190 GOTO 50
```

Befehle Ton SOUND

Direkte Ansprache der Register des PSG (Progr. Sound Generator)

Es gibt verschiedene Möglichkeiten, mit den MSX-Computern Sound zu erzeugen: Sie können mit dem Befehl BEEP einen Signalton abgeben (wie er auch bei einer Fehleranzeige erklingt), mit dem dritten SCREEN-Parameter das Clickgeräusch beim Drücken einer Taste an- bzw. ausschalten sowie mit PLAY Noten in drei Musikkanälen gleichzeitig spielen.

Der SOUND-Befehl hingegen ermöglicht es Ihnen, den PSG (Programmable Sound Generator) direkt anzusprechen. Dabei stehen Ihnen 14 Register zur Verfügung, die wir nun etwas genauer in Ihrer Funktion untersuchen wollen:

Register 0 Eingabemöglichkeit z.B. SOUND 0,200

Mit diesem Register geben Sie an, wie hoch bzw. tief der ausgegebene Ton im Soundkanal A erklingen soll. Voraussetzung dazu ist es allerdings, daß Sie im Register 7 angegeben haben, daß in Kanal A ein Ton erklingen soll und im Register 8 die Lautstärke eingestellt haben. Sie können in Register 0 bis zu 256 (0 bis 255) verschiedene Einstellungen wählen.

Register 1 Eingabemöglichkeit z.B. SOUND 1,10

Mit diesem Register geben Sie ebenfalls an, wie hoch bzw. tief ein Ton in Kanal A erklingen soll. Jedoch weist Register 1 nicht eine derart fein gegliederte Höhenunterscheidung auf, wie dies bei Register 0 der Fall war. Aus diesem Grund können Sie auch lediglich 16 verschiedene Tonhöhenabstufungen (0 bis 15) mit Register 1 vornehmen, jedoch überspringen Sie mit jeder SOUND-Eingabe in Register 1 gleich eine gesamte Oktave.

Man muß Register 0 und 1 als Registerpaar ansehen, wobei mit Register 1 die Grobeinstellung (0 bis 15) und mit Register 0 die Feinabstimmung der Tonhöhe vorgenommen werden kann.

So wie das Registerpaar 0 und 1 (Fein- und Grobeinstellung der Frequenz des Musikkanals A) zusammengehört, verhält es sich auch bei den Registerpaaren 2 und 3 sowie 4 und 5:

Register 2 ist für die Feineinstellung der Frequenz bei Soundkanal B zuständig (Werte 0 bis 255).

Register 3 ist für die Grobeinstellung der Frequenz bei Soundkanal B zuständig (Werte 0 bis 15).

Register 4 ist für die Feineinstellung der Frequenz bei Soundkanal C zuständig (Werte 0 bis 255).

Register 5 ist für die Grobeinstellung der Frequenz bei Soundkanal C zuständig (Werte 0 bis 15).

Sie können, wie bei den Registern zu Soundkanal A bereits angedeutet, nur dann einen Ton erzeugen, wenn:

1) bei Register 7 (Mixregister) der bzw. die entsprechenden Soundkanäle auf Tonwiedergabe eingestellt sind (andere Möglichkeiten: Rauschen, Rauschen und Ton, nichts).

2) bei Register 8 die Lautstärke für Kanal A bzw. bei Register 9 die Lautstärke für Kanal B bzw. bei Register 10 die Lautstärke für Kanal C mit Werten von jeweils zwischen 1 und 15 angegeben sind.

Register 6 Eingabemöglichkeit z.B. SOUND 6,5

Mit Hilfe des SOUND-Befehls können wir nicht nur Töne erzeugen, sondern auch unterschiedliches Rauschen hervorrufen. Zu diesem Zweck muß in Register 7 angegeben sein, welcher Soundkanal Töne und welcher Kanal Rauschen ausgeben soll.

Haben wir uns dazu entschlossen, z.B. Kanal A rauschen zu lassen, können wir mit Register 6 erreichen, daß bis zu 8 verschiedene Rauschzustände wiedergegeben werden (Eingabe 0 bis 7).

Register 7 Eingabemöglichkeit z.B. SOUND &B10101010

Wie Sie sehen, haben wir zur Darstellung der Eingabe bei Register 7 die binäre Schreibweise vorgezogen. Selbstverständlich ist es auch möglich, den im Beispiel angegebenen Wert dezimal oder gar hexadezimal einzugeben; jedoch würde in diesem Fall die Funktion dieses Soundparameters nicht deutlich genug zu erkennen sein.

Durch die Eingabe der drei niederwertigen Bits (in unserem Beispiel '010') wird angegeben, welcher der 3 Soundkanäle zur Tonausgabe bestimmt ist. In unserem Beispiel weist uns somit die '1' darauf hin, daß der Kanal B einen Ton erzeugt, während die zwei Nullen angeben, daß aus Kanal A und C keine Tonausgabe erfolgt.

Die folgenden drei Bits (in unserem Beispiel die Folge '101') gibt an, welche Kanäle ein Rauschen erzeugen. In unserem Beispiel würden also die Kanäle A und C (beide auf '1'=an gesetzt) rauschen, während Kanal B einen Ton abgeben würde (s.o.).

Die zwei hochwertigen Bits müssen in jedem Fall '10' gesetzt sein, ansonsten hat diese Folge keine weitere Bedeutung.

Schauen wir uns noch ein weiteres Beispiel an, das die Bedeutung des siebenten SOUND-Parameters deutlich machen soll:

SOUND 7,&B10100110

In diesem Fall kann nur durch die Soundkanäle B und C ein Ton ausgegeben werden (Folge '110'), während in Kanal C zusätzlich auch noch ein Rauschen erzeugt wird (Folge '100'). Wir können also mit Hilfe des siebenten Soundparameters einen Ton und gleichzeitig ein Rauschen durch einen einzigen Soundkanal ausgeben.

Register 8 Eingabemöglichkeit z.B. SOUND 8,30

Mit diesem Register geben wir an, wie laut die Ton- bzw. die Rauschausgabe in Kanal A erfolgen soll. Wir können Werte zwischen 0 (aus) bis 15 eingeben, wobei sich die Lautstärke zunehmend verstärkt. Geben Sie größere Werte als 15 ein, .bleibt die größtmögliche Lautstärke zwar bestehen, aber nun können Sie gleichzeitig auch noch verfremdete Töne wiedergeben, die Sie mit den SOUND-Registern 11 bis 13 bestimmen. Eine Hüllkurve läßt sich erst dadurch wiedergeben, wenn wir die entsprechende Kanallautstärke (im Fall von Register 8 bezieht sich diese Eingabe auf Kanal A) eingeben.

Register 9 und 10 arbeiten gleich wie Register 8, jedoch beziehen sich die Eingaben auf die entsprechenden anderen Kanäle B und C:

Register 9 ist für die Lautstärke (0 bis 15) bzw. die Hüllkurvenaktivierung (16) von Kanal B zuständig.

Register 10 ist für die Lautstärke (0 bis 15) bzw. die Hüllkurvenaktivierung (16) von Kanal C zuständig.

Wir können also mit folgenden Eingaben bewirken, daß in Kanal A Töne, in Kanal B Rauschen und in Kanal C eine Hüllkurve zu einem vorher gewählten Ton erzeugt werden:

```
SOUND 1,10
SOUND 5,12
SOUND 6,4
SOUND 7,&B10010101
SOUND 8,10
SOUND 9,10
SOUND 10,10
```

Register 11 Eingabemöglichkeit z.B. SOUND 11,11

Die Register 11 und 12 arbeiten ähnlich zusammen, wie dies bereits bei den Registern 0 und 1 der Fall war. Während wir mit Register 11 die Feineinstellung der Hüllkurve vornehmen, können wir mit Register 12 die Grobeinstellung bestimmen. Beide Register üben nur dann eine Funktion aus, wenn zuvor durch Register 13 eine bestimmte Hüllkurvenform bestimmt wurde und mit den Registern 8 bzw. 9 bzw. 10 die Lautstärke auf 16 gesetzt wurde, damit die Ausgabe der Hüllkurven aktiviert ist.

Register 11 und 12 bewirken hierbei, daß bei entsprechend großer Zahleingabe die Frequenz der Hüllkurve zunimmt.

Register 12 dient, wie bereits oben unter Register 11 angeführt, zur groben Hüllkurveneinstellung.

Sowohl bei Register 11 als auch bei Register 12 können Eingaben zwischen 0 und 255 vorgenommen werden, so daß wir insgesamt 65535 Möglichkeiten haben.

Register 13 Eingabemöglichkeit z.B. SOUND 13,4

Es gibt viele Möglichkeiten, einen Ton unterschiedlich darzustellen: So kann der Ton vom Anschlag bis zum Ausklingen gleich laut sein (wie bei einer Autohupe) oder er kann wellenförmig auf- bzw. abschwingen (wie bei einer Sirene) oder ... Mit dem MSX-Soundchip, dem PSG (Programmable Sound Generator) können wir acht verschiedene Hüllkurven erzeugen:

0: Ton ist sofort bei Anschlag voll da und klingt langsam aus

4: Ton steigt bei Anschlag langsam an und verliert sich nach Erreichen der größten Amplitude im Nichts

8: Ton ist anfangs ganz laut, wird leiser und ist dann wieder ganz laut usw.

10: Ton wird leiser, dann wieder lauter, dann wieder leiser usw.

11: Ton wird leiser und ist nach einer kurzen Ruhepause voll da (ohne Ab- bzw. Aufstieg)

12: Ton wird langsam lauter, dann ganz leise, dann wird er wieder langsam lauter usw.

13: Ton wird langsam lauter und behält schließlich die Höchstlautstärke bei

14: Ton wird langsam lauter, dann wieder leiser, dann wieder lauter (ähnlich wie Hüllkurve 10, bei der jedoch die Lautstärke am Anfang voll da ist).

Fassen wir noch einmal kurz zusammen, was die 14 verschiedenen SOUND-Parameter bewirken und wie groß bzw. wie klein die Eingabemöglichkeiten sind:

Register 0: Feineinstellung Frequenz Kanal A (0 bis 255)

Register 1: Grobeinstellung Frequenz Kanal A (0 bis 15)

Register 2: Feineinstellung Frequenz Kanal B (0 bis 255)

Register 3: Grobeinstellung Frequenz Kanal B (0 bis 15)

Register 4: Feineinstellung Frequenz Kanal C (0 bis 255)

Register 5: Grobeinstellung Frequenz Kanal C (0 bis 15)

Register 6: Unterschiedliche Rauscharten (0 bis 7)

Register 7: die kleinsten drei Bits: Ton/Kanal an/aus
die nächsten drei Bits: Rauschen/Kanal an/aus
die höchsten zwei Bits: verbindlich '10'

Register 8: Lautstärke Kanal A (0 bis 15) bzw.
16 = Hüllkurvenaktivierung

Register 9: Lautstärke Kanal B (0 bis 15) bzw.
16 = Hüllkurvenaktivierung

Register 10: Lautstärke Kanal C (0 bis 15) bzw.
16 = Hüllkurvenaktivierung

Register 11: Feineinstellung der Hüllkurvenfrequenz
(0 bis 255)

Register 12: Grobeinstellung der Hüllkurvenfrequenz
(0 bis 255)

Register 13: Hüllkurvenauswahl 0,4,8,10,11,12,13 oder 14

Es reicht nicht aus, wenn wir z.B. nur mit Register 13 die Hüllkurve setzen. Für verschiedene Fälle müssen wir auch unterschiedlich v.

Fall 1: Es soll ein Ton durch Kanal A erklingen:

1) Register 0 für Tonhöhe setzen
2) eventuell Register 1 für Tonhöhe setzen
3) Register 7 so setzen, daß durch Kanal A ein Ton erklingt
4) Register 8 für die Lautstärke einen Wert zwischen 1 und 15 eingeben.

Fall 2: Es soll ein Rauschen durch Kanal B erklingen:

1) Register 6 auf eine Rauschqualität einstellen
2) Register 7 so eingeben, daß durch Kanal B kein Ton aber Rauschen erklingen kann
3) Register 9 für die Lautstärke einen Wert zwischen 1 und 15 eingeben.

Fall 3: Ein auf- und abschwingender Sirenenton soll durch Kanal C erklingen:

1) Register 4 für Tonhöhe setzen
2) eventuell Register 5 für Tonhöhe setzen
3) Register 7 so einegben, daß durch Kanal C ein Ton erklingt
4) Register 10 für die Aktivierung der Hüllkurve 16 eingeben
5) Register 11 und 12 für die Hüllkurvenfrequenz einstellen
6) Register 13 die passende Hüllkurve (höchstwahrscheinlich Nummer 10 oder Numer 14) aussuchen.

Der große Vorteil der SOUND-Eingabe mit Hilfe des SOUND-Befehls ist der, daß ein einmal aktivierter Klang (selbstverständlich auch nach Belieben dreistimmig) so lange weiterklingt, bis wir ihn selbst mit unserem MSX-Computer zum Abbruch zwingen. So ist es z.B. denkbar, daß wir in einem Spiel andauernd Meeresrauschen oder nach Belieben auch Hubschraubergeschnatter erklingen lassen, während das Programm ohne Unterbrechung weiterläuft.

Übrigens:

Sie können mit Register 7 bestimmte Kanäle mit Rauschen belegen. Sollten Sie hiernach versuchen, mit PLAY Töne wiederzugeben, zeigt Ihr MSX-Computer, daß er sich die Eingaben gemerkt hat. Kein Ton erklingt mehr, sondern vielmehr ein Rauschen.

Geben Sie den Befehl BEEP ein oder unterbrechen das Programm auf die Art und Weise, daß ein BEEP-Ton erklingt, sind Ihre bis dahin definierten SOUND-Register samt und sonders gelöscht. Es muß mit der Eingabe wieder von vorn angefangen werden.

Wer bei der Tonausgabe etwas Handfesteres erreichen will, kann sich die Tonfrequenz mit folgender Formel errechnen:

für Kanal A: 124797/256*Register 1+Register 2
für Kanal B: 124797/256*Register 3+Register 4
für Kanal C: 124797/256*Register 5+Register 6

Sie können nur in die 14 SOUND-Register etwas schreiben, aber die Information nicht wieder herauslesen. Eine Teilkontrolle ist mit dem Kommando PLAY(N) möglich: Setzt man für N die Kanalnummer (0 bis 2), erfährt man, ob gerade ein Ton erklingt (wahr=-1) oder aber nicht (falsch=0).

Befehle Sound BEEP

Erzeugen eines 'Aufmerksamkeitstons'

siehe hierzu auch die Programme im Anhang: BEEP 1
BEEP 2
BEEP 3

Gebrauchen wir diesen Befehl, läuft es so manchem Programmierer schaurig den Rücken herunter, denn der hierdurch erzeugte Ton entspricht genau dem, der auftritt, wenn der Computer im Programm auf einen Fehler trifft und eine Meldung auf dem Bildschirm ausgibt (z.B. Syntax error in 10).

Sie können dies ausprobieren, indem Sie wirre Zeichen (ohne Zeilennummer) eingeben und anschließend die ENTER-Taste drücken.

Wir können den gleichen 'Aufmerksamkeitston' auch auf folgende Arten erzeugen:

1) PRINT CHR$(7)

2) Drücken der Tasten CTRL und G

3) PLAY "L64 M255 O6 S1 T120 V8 E"

4) folgendes Programm:

siehe Kurzlisting BEEP 1 im Anhang

Nun haben wir gleich sechs Arten (inklusive der Errormeldung und des direkt eingegebenen Befehls BEEP) kennengelernt, wie wir diesen hellklingenden kurzen Ton erzeugen können. Ehrlich gesagt, ein bißchen zu viel des Guten, werden Sie sicherlich und auch ganz richtig feststellen. Jedoch haben die ausgegebenen BEEP-Töne, obwohl sie gleich klingen, nicht alle die gleiche Bedeutung! Probieren Sie es einmal aus:

siehe Kurzlisting BEEP 2 im Anhang

Setzen Sie nacheinander die anderen BEEP-Erzeuger (PRINT CHR$(7), den PLAY-Befehl und das SOUND Programm) in Zeile 20 ein (bzw. das Soundprogramm in die Zeilen 20 bis 25) und versuchen Sie, einen Unterschied festzustellen.

Sicherlich ist es Ihnen aufgefallen: Wenn wir BEEP oder PRINT CHR$(7) eingeben, werden die Tonregister beim PLAY- und beim SOUND-Befehl (in diesem Fall die angewählte Oktave) zurückgesetzt. Verwenden wir aber den künstlich erzeugten BEEP-Ton mit Hilfe des PLAY-Befehls oder des SOUND-Programms, hat sich Ihr MSX-Computer noch die alten Einstellungen gemerkt d.h. hier werden die Tonregister so belassen, wie wir dieselbigen ursprünglich eingestellt hatten. Aus diesem Grund ist es ratsam, wenn wir Musikeinstellungen in unseren Programmen verwenden, den PLAY-Befehl oder das SOUND-Programm für 'Aufmerksakeitstöne' heranziehen. So haben wir weiterhin all' die Soundregister direkt unter Kontrolle.

Das plötzliche und manchmal unerwartete Auftreten des Error-Beeps hingegen läßt sich dadurch eliminieren, wenn wir für die zu erwartenden Fehlermeldungen zu Beginn unseres Programms eine entsprechende ON ERROR-Interruptsteuerung einbauen.

Noch ein Hinweis: Ich weiß nicht, ob Sie meine Wortwahl richtig verstanden haben, als ich von 'Aufmerksamkeitstönen' sprach. Um dies hier zu verdeutlichen, ein praktisches Beispiel mit dem BEEP-Befehl:

siehe Kurzlisting BEEP 3 im Anhang

Wenn Sie bei diesem Programm entgegen der Anweisung eine Zahl eingeben, die kleiner als 0 oder größer als 50 ist, erklingt der BEEP-Ton, also ein Ton, der Sie auf den Fehler aufmerksam machen soll (ein 'Aufmerksamkeitston').

BEEP 1

```
        -*-*-*-*-*-
10 SOUND Ø,85
20 SOUND 1,Ø
30 SOUND 7,8
40 SOUND 8,7
50 FOR N=1 TO 1Ø
60 NEXT N
70 SOUND 7,9
```

BEEP 2

```
        -*-*-*-*-*-
10 PLAY "O6 CEG"
20 FOR N=1 TO 1ØØØ
30 NEXT N
40 BEEP
50 PLAY "CEG"
```

BEEP 3

```
        -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN Ø
30 PRINT "Geben Sie eine Zahl"
40 INPUT "zwischen Ø und 5Ø ein ";A
50 IF A<Ø OR A>5Ø THEN BEEP:GOTO 3Ø
60 PRINT "Gut gemacht!"
70 GOTO 3Ø
```

Befehle Grafik STICK

Abfrage der Joystickrichtung und der Pfeiltasten

siehe hierzu auch die Programme im Anhang: STICK 1
STICK 2
STICK 3
FRETY
MISTER MINER

Wie uns der Name dieses Befehls bereits andeutet, handelt es sich hier wieder um eine Abfrage, die mit einem möglicherweise an eine der zwei Joystickbuchsen Ihres MSX-Computers angeschlossenen Steuergerät zusammenhängt.

Wir hatten diesbezüglich bereits die Kommandos STRIG (Feuerknopf gedrückt ja/nein), PAD (ein angeschlossenen Malpad betreffend) und PDL (ein angeschlossenes Paddle betreffend) kennengelernt.

Mit dem MSX-Befehl STICK sind wir nun in der Lage, bei einem angeschlossenen Joystick die gewählte Richtung abzufragen. Da Sie an Ihren MSX-Computer nicht nur einen Joystick anschließen können, muß der STICK-Befehl für seinen speziellen Nutzen noch durch einen Parameter eingegrenzt werden:

PRINT STICK(0)=Abfrage der Pfeiltasten auf der Tastatur
PRINT STICK(1)=Abfrage des Joysticks in Port 1
PRINT STICK(2)=Abfrage des Joysticks in Port 2

Es ist mit Hilfe dieser drei Abfragen somit möglich, bis zu drei Spieler gleichzeitig an einem Spiel teilhaben zu lassen: Während zwei Spieler ihren Joystickhebel in die passende Richtung schwenken, sitzt Spieler drei an der Tastatur und benutzt zur Richtungsänderung halt das Pfeiltastenkreuz.

Erinnern wir uns: Auch die Feuerknopfabfragen mit Hilfe der Kommandos ON STRIG und STRIG kann man nicht nur auf zwei angeschlossene Joysticks, sondern gleichzeitig auch auf die gedrückte bzw. nicht gedrückte Leertaste beziehen. Also wahrlich: Ein Spiel mit drei Personen ist samt aller möglichen Steuerungstätigkeiten mit Ihrem MSX-Computer möglich - noch viel

mehr: mit ON STRIG können Sie gar interruptgesteuert erst dann das Programm unterbrechen, wenn wahrlich ein Spieler die Feuertaste bzw. die Leertaste betätigt hat.

Bevor wir uns an Beispielen die Bedeutung des STICK-Befehls praktisch vor Augen führen, noch schnell ein Blick auf die Bedeutung der Resultate, die uns die STICK-Abfrage anzeigt: Schauen wir uns dafür die Richtungen und deren entsprechende Zahlwerte an:

0 = keine Richtung angewählt (Ausgangs = Ruhezustand, Joystick in Mittenposition)

1 = Bewegung nach rechts oben oder Himmelsrichtung NO (auf der Tastatur bitte Pfeil nach oben und Pfeil nach rechts gleichzeitig drücken)

2 = Bewegung nach rechts oder Himmelsrichtung O

3 = Bewegung nach rechts unten oder Himmelsrichtung SO (auf der Tastatur bitte Pfeil nach rechts und Pfeil nach unten gleichzeitig drücken)

4 = Bewegung nach unten oder Himmelsrichtung S

5 = Bewegung nach links unten oder Himmelsrichtung SW (auf der Tastatur bitte Pfeil nach unten und Pfeil nach links gleichzeitig drücken)

6 = Bewegung nach links oder Himmelsrichtung W

7 = Bewegung nach links oben oder Himmelsrichtung NW (auf der Tastatur bitte Pfeil nach links und Pfeil nach oben gleichzeitig drücken)

8 = Bewegung nach oben oder Himmelsrichtung N

Wie bereits in obiger Aufstellung im Einzelfall darauf hingewiesen wurde, können wir mit der Tastatur die Zwischenwerte nur so ansteuern, indem zwei Richtungstasten gleichzeitig betätigt

werden: soll es nach rechts oben gehen, drücken Sie bitte die Pfeiltasten nach oben und nach rechts zur gleichen Zeit.

Würden Sie hingegen bei der Tastatur aus Versehen zwei Tasten zusammen drücken, die entgegengesetzte Himmelsrichtungen angeben, würde auch dies Ihr MSX-Computer verzeihen; er würde dies gar ignorieren: Wird z.B. die rechte und linke Pfeiltaste gleichzeitig gedrückt, ist das Ergebnis = 0 und gar eine dritte gedrückte Pfeiltaste z.B. die Pfeiltaste nach unten wird so interpretiert, als wären Rechts- und Links-Taste gar nicht gedrückt worden. Die Ziffernausgabe würde also in letztgenanntem Fall '5' für unseren Tastaturspieler lauten.

Kommen wir nun zu zwei kurzen Beispielen, die uns die Leistungsfähigkeit des STICK-Befehls praktisch vor Augen führen:

1) Malprogramm mit Joystick oder Tastatur

Hierzu noch drei Hinweise:

1. Da Sie für die Zahlen 0 (Tastatur), 1 (Joystick 1) und 2 (Joystick 2) wie in diesem Beispiel gezeigt auch eine Variable nehmen können, können Sie in Ihren Programmen ganz einfach dem User die Absicht überlassen, ob er dieses Spiel nun mit Tastatur oder mit Joystick spielen will.

2. Erscheint Ihnen die Bewegung des Bild-Malpunktes zu langsam, können Sie ganz einfach die Zahlen hinter dem PSET STEP-Befehl vergrößern (z.B. für Himmelsrichtung NO: PSET STEP (+2,-2).

3. Obgleich es Ihrem MSX-Computer egal ist, ob Sie sich nun in seiner Anzeigematrix (256*192 Punkte) bewegen oder nicht, in jedem Fall sollten Sie in Ihren Programmen die Extremkoordinaten wieder auf 0 setzen, damit eine realistische Abbildung auf dem Bildschirm stattfindet und Ihnen der Grafikcursor nicht zu weit von der sichtbaren Bildfläche verschwindet.

siehe Kurzlisting STICK 1 im Anhang

2) Joystick-Richtungsabfrage

Bis auf wenige Ausnahmen sind heutzutage sämtliche Joysticks bruchgefährdet. Damit Sie nicht den Händler irremachen (manchmal funktioniert gar nichts, manchmal aber auch doch) können Sie mit dem folgenden Programm mit Hilfe einer Grafik genauestens untersuchen, welche Richtungen bei Ihrem Joystick einwandfrei funktionieren und welche nicht:

siehe Kuzlisting STICK 2 im Anhang

Selbstverständlich können Sie dieses Programm auch zur Abfrage der Funktionstüchtigkeit der Pfeiltasten Ihres MSX-Computers heranziehen. Dies dürfte aber als zu aufwendig gelten, denn schließlich kann diese Überprüfung direkt ausgeführt werden (Pfeiltasten als Editor).

Noch ein Hinweis zur Tastaturabfrage mit PRINT STICK(0): Da hinter jeder gedrückten Taste Ihres MSX-Computers ein Zahlencode verborgen liegt, den wir mit PRINT ASC(A$) auch abfragen können, gibt es für Sie zur Abfrage der Pfeiltasten gleich zwei Möglichkeiten:

1. Abfrage über PRINT STICK(0) (s.o.)

2. Abfrage des gedrückten Tastaturcodes, so wie es uns im folgenden Programm gezeigt wird:

siehe Kurzlisting STICK 3 im Anhang

Die Pfeiltasten tragen folgende Codes:

rechts	= CHR$(28)	links	= CHR$(29)
oben	= CHR$(30)	unten	= CHR$(31)

Jedoch gibt es ein Problem mit dem zuletzt eingegebenen Programm: Wollen Sie die Zwischenrichtungen abfragen, bekommen Sie als Ergebnis immer abwechselnd zwei Zahlwerte, jeweils für die Richtungen, aus denen sich die Zwischenrichtung zusammensetzt.

STICK 1

```
             -*-*-*-*-*-
10 SCREEN Ø
20 COLOR 1,15
30 WIDTH 4Ø
40 PRINT "Tastatur         = Ø
          Joystickport 1 = 1
          Joystickport 2 = 2"
50 PRINT
60 INPUT "Ihre Eingabe bitte (Ø bis 2) "
;A
70 IF A<Ø OR A>2 THEN RUN
80 SCREEN 2
90 PSET(128,96)
100 B=STICK(A)
110 IF B=1 THEN PSET STEP(Ø,-1)
120 IF B=2 THEN PSET STEP(+1,-1)
130 IF B=3 THEN PSET STEP(+1,Ø)
140 IF B=4 THEN PSET STEP(+1,+1)
150 IF B=5 THEN PSET STEP(Ø,+1)
160 IF B=6 THEN PSET STEP(-1,+1)
170 IF B=7 THEN PSET STEP(-1,Ø)
180 IF B=8 THEN PSET STEP(-1,-1)
190 GOTO 1ØØ
```

STICK 2

```
             -*-*-*-*-*-
1Ø SCREEN Ø
2Ø COLOR 1,15
3Ø WIDTH 4Ø
4Ø PRINT "Tastatur         = Ø
          Joystickport 1 = 1
          Joystickport 2 = 2"
5Ø PRINT
6Ø INPUT "Ihre Eingabe bitte (Ø bis 2) "
;A
7Ø IF A<Ø OR A>2 THEN RUN
8Ø ON STRIG GOSUB 24Ø,24Ø,24Ø,24Ø,24Ø
9Ø FOR N=Ø TO 4
1ØØ STRIG(N) ON
11Ø NEXT N
12Ø SCREEN 2
13Ø B=STICK(A)
14Ø IF B=Ø THEN GOTO 13Ø ELSE:PSET (128,
96)
15Ø IF B=1 THEN DRAW "U16 N F8 G8"
16Ø IF B=2 THEN DRAW "E16 N D8 L8"
17Ø IF B=3 THEN DRAW "R16 N H8 G8"
18Ø IF B=4 THEN DRAW "F16 N U8 L8"
19Ø IF B=5 THEN DRAW "D16 N H8 E8"
2ØØ IF B=6 THEN DRAW "G16 N U8 R8"
```

```
210 IF B=7 THEN DRAW "L16 N E8 F8"
220 IF B=8 THEN DRAW "H16 N D8 R8"
230 GOTO 130
240 CLS
250 RETURN
```

STICK 3

```
            -*-*-*-*-*-
10 SCREEN 0
20 COLOR 1,15
30 WIDTH 40
40 PRINT "Bitte die Pfeiltasten testen"
50 A$=INKEY$
60 IF A$="" THEN GOTO 50
70 LOCATE 10,10
80 IF ASC(A$)=28 THEN PRINT "rechts"
90 IF ASC(A$)=29 THEN PRINT "links "
100 IF ASC(A$)=30 THEN PRINT "oben  "
110 IF ASC(A$)=31 THEN PRINT "unten "
120 GOTO 50
```

Befehle Grafik STRIG

Abfrage von bis zu vier Joystickknöpfen und der Leertaste

siehe hierzu auch die Programme im Anhang: STRIG 1
STRIG 2
FRETY
MISTER MINER

Erneut ein Befehl, der seinen Nutzen bzw. seine Informationen u.a. aus den Joystickports und/oder der Tastatur bezieht. Neben STICK (Richtungsabfrage), PAD (Maltablettabfrage) und PDL (Paddleabfrage) beschäftigt sich das Kommando STRIG nicht mit einer Richtungsabfrage, sondern vielmehr 'nur' mit einer gedrückt/nicht gedrückt-Abfrage.

Überprüft wird, ob ein Joystickknopf oder/und die Leertaste auf der Tastatur gedrückt wurden. Wir bedienen uns bei der Abfragekontrolle der luxuriösen Interruptabfrage mit ON STRIG GOSUB. So ist es nicht ununterbrochen nötig, in langwierigen Programmschleifen sehnsüchtig darauf zu warten, wann die Leertaste oder ein Feuerknopf gedrückt wurden.

Es genügt nicht, einfach PRINT STRIG einzugeben. Vielmehr müssen wir uns bei der Abfrage genauer festlegen:

PRINT STRIG(0) = wurde die Leertaste gedrückt?

PRINT STRIG(1) = wurde Feuerknopf 1 von Joystick 1 gedrückt?

PRINT STRIG(2) = wurde Feuerknopf 1 von Joystick 2 gedrückt?

PRINT STRIG(3) = wurde Feuerknopf 2 von Joystick 1 gedrückt?

PRINT STRIG(4) = wurde Feuerknopf 2 von Joystick 2 gedrückt?

Hierbei ist folgendes zu beachten: Die Abfrage des gedrückten bzw. nicht gedrückten Feuerknopfes 2 sollten Sie nur vornehmen, wenn Sie wahrlich einen Joystick besitzen, der der MSX-Norm entspricht und zwei unterschiedlich abzufragende Feuerknöpfe besitzt. Erstellen Sie ein Spiel, das für den späteren Verkauf bestimmt ist, sollten Sie zumindest die Möglichkeit in einem Anfangsmenü schaffen, daß auch mit einem normalen Joystick (nur 'ein' abfragbarer Joystickknopf) der Spielinhalt vollkommen ausgereizt werden kann. Denn Sie müssen immer daran denken: Ein MSX-Computerbesitzer ist noch lange nicht Inhaber eines dazu 100 prozentig passenden Joysticks mit jeweils zwei unterschiedlich abfragbaren Feuerknöpfen. Ist Ihr Spiel gar für drei Spieler geeignet, muß der Gerechtigkeit halber die zweite Joystickfeuerknopfabfrage mit STRIG fallengelassen werden, denn bei der Tastatur kann in jedem Fall nur eine Feuertaste = die Leertaste abgefragt werden.

Ich spach vorhin bereits auf die enge Verwandtschaft der MSX-Kommandos ON STRIG GOSUB und PRINT STRIG an. Schauen wir uns deshalb einmal in einem kleinen Programm die Wirkung dieser zwei Befehle etwas genauer an:

siehe Kurzlisting STRIG 1 im Anhang

Zwei Sonderheiten dürften Ihnen in diesem Programm aufgefallebn sein:

1. Es wird noch ein weiterer STRIG-Befehl verwendet, der die Interruptabfrage mit ON STRIG an- und ausschaltet:

STRIG ON bzw.
STRIG OFF

2. Die Abfrage, welcher Feuerknopf gedrückt wurde, läßt sich sehr einfach durch die wahr (-1) = Feuerknopf bzw. Leertaste gedrückt oder falsch (0)-Aussage = Feuerknopf bzw. Leertaste nicht gedrückt bewerkstelligen. IF STRIG(0) THEN bedeutet: wenn der erste Teil der Frage richtig beantwortet wurde, dann führe auch das nachfolgende Kommando aus, andererseits gehe in die nächste Programmzeile.

Noch ein Hinweis: Sie sollten die Abfrage der Leertaste (gedrückt oder nicht gedrückt) mit Hilfe des STRIG-Befehls nicht überbewerten. Wie sich im folgenden Programm zeigt, kann die Abfrage der Leertaste auch über die Befehlskombination

IF ASC(A$)=32

bewerkstelligt werden.

siehe Kurzlisting STRIG 2 im Anhang

Jedoch können Sie bei diesem Programm die Interruptabfrage mit ON STRIG GOSUB nun nicht mehr für die Abfrage Leertaste gedrückt ja/nein verwenden.

STRIG 1

-*-*-*-*-*-

```
10 SCREEN 0
20 ON STRIG GOSUB 70,90,110,130,150
30 FOR N=0 TO 4:STRIG(N) ON
40 STRIG(N) ON
50 NEXT N
60 GOTO 60
70 PRINT "Leertaste gedrueckt"
80 RETURN
90 PRINT "Joystick 1, Feuer 1 gedrueckt"
100 RETURN
110 PRINT "Joystick 2, Feuer 1 gedrueckt
"
120 RETURN
130 PRINT "Joystick 1, Feuer 2 gedrueckt
"
140 RETURN
150 PRINT "Joystick 2, Feuer 2 gedrueckt
"
160 RETURN
```

STRIG 2

-*-*-*-*-*-

```
10 SCREEN 0
20 PRINT "Bitte die Leertaste druecken"
30 PRINT
40 A$=INKEY$
50 IF A$="" THEN GOTO 40
60 IF ASC(A$)=32 THEN PRINT "Leertaste g
edrueckt"
70 GOTO 40
```

Befehle Grafik PAD
Koordinatenabfrage des an den Joystickport angeschlossenen Pad

siehe hierzu auch die Programme im Anhang: PAD 1
PAD 2
PAD 3

Im Gegensatz zu den meisten Homecomputern verfügen die MSX-Computer nicht nur über 2 Joystickanschlüsse, sondern sie besitzen auch ein BASIC-Befehlsspektrum ohnegleichen, mit dem sie den Gebrauch dieser Anschlüsse besonders gut nützen können.

Einer dieser Joystickanschlußbefehle heißt PAD und bedeutet so viel wie: welche Eingaben gehen im Augenblick am angeschlossenen PAD ein. Diese Abfrage geschieht über eine dimensionierte BASIC-Variable, die ebenfalls den Namen PAD trägt.

Bevor wir uns jedoch eingehender mit diesem BASIC-Befehl auseinandersetzen, hier eine kurze Übersicht, was ein PAD überhaupt ist und wo man dasselbige sinnvoll einsetzen kann.

Das PAD ist im Grunde genommen eine Art Tablett, dessen Hauptfläche aus einem druckempfindlichen Material besteht. Wir können auf dieser druckempfindlichen Fläche mit einem spitzen Gegenstand Signale auslösen, die über den Joystickanschluß Mitteilung darüber abgeben, wo der Druck erfolgt ist. Übersetzt heißt das: Wir müssen uns die PAD-Fläche vorstellen, wie unseren Computerbildschirmen. Drücken wir links oben mit einem spitzen Gegenstand auf das PAD, so registriert der MSX-Computer dieses Signal an der entsprechenden Stelle und verarbeitet die Dateieingabe wie auf einem Koordinatensystem.

Einerseits stellt unser MSX-Computer also fest, daß wir durch Andrücken des spitzen Gegenstandes eine Aktion übermitteln wollen: er kann aber auch genau feststellen, an welcher Koordinatenposition dieser Andruck erfolgte. Außerdem befindet sich auf dem PAD noch eine zusätzliche Drucktaste (ähnlich dem Joystickfeuerknopf), die ebenfalls beim Niederdrücken im Computer eine entsprechende Reaktion erzeugt. Beide Eingaben werden registriert und können mit dem MSX-Befehl PAD abgefragt werden.

Verwendung findet das PAD, indem Programme im Computer derart abgestimmt sind (s.u.), daß auf Druck in der linken oberen Ecke auf dem PAD auch ein Punkt links oben auf dem Bildschirm angezeigt wird, entsprechend unten rechts auch diese Aktion unten rechts auf dem Bildschirm angezeigt wird. Also könnte man mit Hilfe des spitzen Gegenstandes auf dem PAD zeichnen und diese Zeichnung würde parallel dazu auch auf dem Bildschirm erscheinen.

Eine ideale Anwendungsmöglichkeit, die zudem Nutzen bringt, wäre folgende: Sie geben Ihrem MSX-Computer einen bestimmten Maßstab ein, indem Sie für eine bestimmte Länge einer Linie die entsprechende Kilometer-Angabe dazuliefern. Nun legen Sie einen Kartenausschnitt mit entsprechendem Maßstab auf das PAD und ... siehe da: die eingezeichnete Linien werden Ihnen vom Computer automatisch ausgerechnet.

Um die sinnvolle Anwendung des zusätzlichen PAD-Knopfes aufzuzeigen, könnte man denselbigen als Umschalter z.B. zwischen zwei verschiedenen Bildern/Maßstäben oder zwischen Bild und Helpmenüs ansehen und verstehen. Genausogut wäre es möglich, mit Hilfe des PAD-Knopfes die Farbe der Zeichenfarbe zu ändern usw. Vergessen wir bei aller Begeisterung für das PAD aber nicht, daß es vorwiegend zur Eingabe von Zeichnungen und der Knopf z.B. zum Umschalten vom Menü zum Bild (und umgekehrt) gebracht wird.

Unser MSX-Computer verfügt leider noch nicht von vornherein über ein eingebautes PAD-Überwachungsprogramm, so daß wir uns dieses selbst schreiben müssen. Schauen wir uns dazu die PAD-Variable ein klein wenig genauer an: wir können über die Befehlskombination PRINT PAD(N) die Variablenbewegung des PAD-Befehls abfragen. Hierbei gilt folgende Zuordnung:

PAD(0) bis PAD(3): Abfrage für Joystickport 1
PAD(4) bis PAD(7): Abfrage für Joystickport 2

Im einzelnen haben die PAD-Variablen folgende Bedeutung:

0 bzw. 4: Lesen des angeschlossenen PADs
1 bzw. 5: Abfrage der X-Koordinate
2 bzw. 6: Abfrage der Y-Koordinate
3 bzw. 7: Lesen des zusätzlichen PAD-Knopfes (s.o.)

Es reicht in keinem Fall aus, nur die PAD-Variable 0 bzw. 4 abzufragen, denn diese PAD-Variable zeigt nur an, ob etwas auf dem PAD passiert (Aussage wahr = -1), nicht aber, wie und wo es sich ereignet.

Wir können PAD(0) bzw. PAD(4) als Einschalter zum PAD-Lesen verstehen, denn ist PAD(0) bzw. PAD(4) -1, ändert sich zur gleichen Zeit mindestens eine der Variablen zur X-Y-Koordinateneingabe (PAD(1) bis PAD(6)).

Fragen wir PAD(0) bzw. PAD(4) ab, wird uns bei gleichzeitigem Zeichnen auf dem PAD (Andrücken eines spitzen Gegenstandes auf das Tablett) die Meldung "wahr = -1" augegeben. Geschieht dies (PRINT PAD(0): Ergebnis = -1) werden PAD(1) und PAD(2) bzw. PAD(5) und PAD(6) mit den aktuellen X,Y-Werten belegt z.B.:

siehe Kurzlisting PAD 1 im Anhang

Bis zum nächsten Andrücken des spitzen Gegenstandes auf dem PAD (falls überhaupt gedrückt - siehe Zeile 40) werden uns die zuletzt eingegebenen Koordinaten auf dem Bildschirm angezeigt z.B. wenn wir an die Mitte des PADs (128,96) mit einem spitzen Gegenstand aufdrücken, passiert bei PAD(0) bis PAD(2) folgendes:

1) PAD(0) hat den Wert -1 (wahr), so lange der Stift angedrückt wird

2) PAD(1) nimmt den aktuellen X-Wert an (Anzeige dieses Werts durch PRINT PAD(1))

3) PAD(2) nimmt den aktuellen Y-Wert an (Anzeige dieses Wertes durch PRINT PAD(2))

Das gleiche gilt übrigens auch, wenn wir unser PAD in Joystickbuchse 2 stecken. In diesem Fall müssen wir unsere Abfrage über die Variablen PAD(4) (ist -1 = wahr, wenn wir mit einem spitzen Gegenstand überhaupt die druckempfindliche Fläche des PADs berühren), sowie PAD(5) und PAD(6) (X- bzw. Y-Koordiate bei der Ansprache über das PAD) durchführen.

PAD(3) für Joystickbuchse 1 sowie PAD(7) für Joystickbuchse 2 gibt uns Auskunft darüber, ob der zusätzliche Knopf auf dem PAD gedrückt wurde (ja d.h. Aussage ist wahr = -1). Wie bereits vorher angedeutet, geben wir hiermit an, daß z.B. in ein anderes Bild umgeschaltet werden soll. wollen.

Schauen wir uns das in folgendem Beispielprogramm mal etwas genauer an:

siehe Kurzlisting PAD 2 im Anhang

Zeile 30: So lange der spitze Stift nicht auf die druckempfindliche Schicht des PADs aufgesetzt wird, verharrt das Programm in Zeile 30 hängen (PAD(0) = 0 = Aussage falsch), andernfalls (PAD(0) = -1 Aussage wahr) springt das Programm weiter in Zeile 40

Zeile 40: Die Koordinaten von X und Y, an denen der Stift das PAD berührte, werden auf dem Bildschirm ausgegeben (z.B. beim Mittelpunkt 128 für X (PAD(1)) und 96 für Y (PAD(2)))

Zeile 50: Nun wird so lange abgewartet, bis der PAD-Knopf gedrückt wurde (PAD(3) = wahr = -1)

Zeile 60: Das Programm springt zurück an den Programmstart

Vorhin haben wir über die Awendung des PADs als Zeichengerät gesprochen. Hier nun ein Beispiel für diesen Zweck:

siehe Kurzlisting PAD 3 im Anhang

Das Programm ist nicht daauf angewiesen, bei jedem Durchlauf abzufragen, ob die druckempfindliche Fläche auch wirklich berührt wurde, weil immer von neuem der aktuelle Grafikcursor gesetzt wird (Zeile 60). Das Schwergewicht liegt hier vielmehr auf der Abfrage des zusäzlichen PAD-Knopfes, der zwischen Vordergrund (schwarz) und Hintergrundfarbe (weiß) je nach Tastendruck wechselt. Mit etwas Geduld können Sie somit Bilder direkt auf dem PAD malen und auf den Bildschirm direkt übertragen. Es ist sogar möglich, bereits bemalte Flächen durch Umstellen auf die Hintergrundfarbe (bitte PAD-Knopf drücken) durch Übermalen mit der Hintergrundfarbe wieder am Bildschirm zu löschen.

Sie können die angesprochene X-Y-Koordinate des PADs auch direkt über folgende Speicherstellen abfragen:

PAD X-Koordinate: &HFC9C
PAD Y-Koordinate: &HFC9D

PAD 1

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 0
30 A=PAD(0)
40 IF A THEN GOTO 50 ELSE GOTO 30
50 PRINT PAD(1)
60 PRINT PAD(2)
70 GOTO 30
```

PAD 2

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 0
30 IF PAD(0)=0 THEN GOTO 30
40 PRINT PAD(1),PAD)(2)
50 IF PAD(3)=0 THEN GOTO 50
60 GOTO 30
```

PAD 3

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 0
30 IF PAD(3)=-1 THEN IF A=2 THEN A=1 ELS
E A=2
40 IF A=2 THEN COLOR 15
50 IF A=1 THEN COLOR 1
60 PSET(PAD(1),PAD(2))
70 GOTO 30
```

Befehle   Grafik   PDL

Abfrage des an die Joystickbuchse angeschlossenen Paddles

siehe hierzu auch die Programme im Anhang: PDL 1
PDL 2

Wir haben eine große Anzahl von Befehlen, die uns das MSX-BASIC bietet, um die zwei Joystickports für verschiedene angeschlossene Geräte zu nutzen. Mit PAD fragen wir X- und Y-Koordinate des angeschlossenen Grafiktabletts ab, mit STRIG die Joystickrichtungen sowie mit STICK den Druck-Nichtdruck des auf ihm befindlichen Feuerknopfes. Mit PDL wird ein weiteres angeschlossenes Steuerinstrument abgefragt und zwar das PaDdLe.

Während beim Joystick die acht möglichen Richtungen abgefragt werden können, ist dies beim Grafiktablett ein angewählter Punkt im Koordinatenstystem auf 256*192 Punkten. Die Steuerung des PaDdLes geschieht jedoch nur eindimensional (entweder von rechts nach links oder von oben nach unten) und zwar in dem Wertbereich von 0 bis 255.

Das PaDdLe ist ein Steuerinstrument, das uns die Möglichkeit mittels eines Drehknopfes bietet, Werte zwischen 0 und 255 einzugeben. Beispielsweise sei an ein Spiel erinnert, in dem sich folgende Handlung am besten mit einem PaDdLe steuern läßt:

Wir befinden uns im Zirkus und müssen eine Wippe am Boden zwischen linker und rechter Bildschirmseite hin- und herbewegen. Aus der Zirkuskuppel springt ein Männchen herab und Sie versuchen nun, ihm die Wippe so unterzuschieben, daß sein Körper nach dem Auftreffen eine möglichst große Sprunghöhe erreicht (je nachdem, ob er die hochgestellte Wippenseite mehr am Ende = größere Hebelwirkung oder mehr zum Drehpunkt hin trifft).

Würden wir dieses Spiel für unseren MSX-Computer schreiben wollen, reicht folgendes Unterprogramm zur Steuerung mittels eines angeschlossenen PaDdLes aus:

siehe Kurzlisting PDL 1 im Anhang

Je nach Drehrichtung des PaDdLes werden nun Werte zwischen 0 und 255 erzeugt, die die direkte Steuerung unseres Sprites in der Horizontalen auf dem Bildschirm ermöglichen.

Also könnte man sagen, daß der Befehl PDL eigentlich kein Kommando sondern vielmehr eine dimensionierte Variable darstellt. Diese Variable erfährt jedoch durchgehend eine Aktualisierung, die der augenblicklichen Position des PaDdLe-Drehknopfs entspricht.

Es reicht nicht allein aus, die Variable PDL abzufragen, wir müssen ihr einen Wert geben, wobei für die zwei vorhandenen Joystick- bzw. PaDdLeanschlüsse folgende Regel gilt:

Parameter geradzahlig (2 bis 12) z.B. PDL(2) oder PDL(4):
Abfrage von Joystickbuchse 2

Parameter ungeradzahlig (1 bis 11) z.B. PDL(1) oder PDL(3):
Abfrage von Joystickbuchse 1

Obgleich wir hier nur die Werte 1 und 2 gebrauchen, ist es auch möglich, anstattdessen z.B. 3 und 4 zu nehmen, aber bitte nicht größere Werte als 12 bzw. kleinere Werte als 1, weil dann Ihr MSX-Computer mit der Fehlermeldung "Illegal function call" den Programmablauf unterbricht.

Noch drei Hinweise:

1) Die Abfrage des Joystickports mit PDL braucht ungefähr 0.01 Sekunden. Dies ist eine recht lange Zeit für den Gebrauch eines MSX-BASIC-Kommandos. Sie können somit diese Variablenabfrage auch als Zeitschleife in Ihren Programmen einsetzen.

2) Ist kein PaDdLe an Ihren MSX-Computer angeschlossen, ist die Ausgabe erfolgt bei Abfrage der PDL-Variablen 255.

3) Da Sie bei Gebrauch eines PaDdLes immer an der zuletzt angesteuerten Bildschirmposition verharren und diese auch wieder von neuem direkt abfragbar ist (beim Joystick schnellt der Steuerhebel wenn wir ihn loslassen direkt in seine Mittelposition zurück bzw. beim Grafiktablett wird der angesteuerte Punkt nur so lange gespeichert, wie wir entsprechend unseren spitzen Gegenstand aufdrücken), können Sie die Abfrage von 2 angeschlossenen PaDdLes auch direkt für ein Zeichengerät nutzen:

siehe Kurzlisting PDL 2 im Anhang

Aber Vorsicht! Obgleich die Y-Ausdehnung auf dem Bildschirm "nur" von 0 bis 191 geht, werden Ihnen bei Abfrage des PDL(2) Werte von 0 bis 255 ausgegeben.

PDL 1

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 2
30 SPRITE$(1)=STRING$(4,0)+CHR$(255)+CHR
$(255)+CHR$(24)+CHR$(24)
40 PUT SPRITE 1,(PDL(1),180)
50 GOTO 40
```

PDL 2

-*-*-*-*-*-

```
10 COLOR 1,15
20 SCREEN 2
30 PSET(PDL(1),PDL(2))
40 GOTO 30
```

An-, Ausstellen und Unterbrechen der Spritekollisionskontrolle

Dieses BASIC-Kommando arbeitet direkt mit dem Befehl ON SPRITE GOSUB zusammen. Hierbei handelt es sich um eine sogenannte Interruptsteuerung d.h. während das BASIC-Programm läuft wird alle ... Zeiteinheiten nachgeprüft (ohne den eigentlichen Programmablauf zu unterbrechen), ob das zu untersuchende Ergebnis eingetreten ist.

Im Fall des ON SPRITE-Befehls würde das heißen: Sind irgendwo auf dem Bildschirm 2 Sprites zusammengestoßen? Ist dies der Fall, wird in das zum Programmbeginn mit ON SPRITE GOSUB angegebene Unterprogramm verzweigt.

Beispiele für eine sinnvolle Anwendung dieser 'Spritekollisionsüberprüfung' finden sich vor allen Dingen im Spielbereich:

z.B. zwei Raumschiffe stoßen zusammen. Im Unterprogramm wird an der entsprechenden Bildschirmposition eine Explosion dargestellt.

oder ein Spieler schießt umherfliegende Gegenstände ab. Sobald seine Munition (ein Sprite) mit einem Gegenstand (ein zweites Sprite) zusammentrifft, verzweigt der Computer in ein Unterprogramm, erhöht das Punktekonto des Spielers um 1 und läßt den abgetroffenen Gegenstand vom Bildschirm verschwinden.

Wofür aber ist dann noch der SPRITE ON/OFF/STOP-Befehl zu gebrauchen? Hiermit können Sie den vorher definierten Sprung ins Unterprogramm - die Zeilennummer dieses Unterprogramms wird ja im ON SPRITE GOSUB-Befehl angegeben - je nach Situation unterbinden (mit SPRITE OFF), wieder neu starten lassen (mit SPRITE ON) oder erst zu einem späteren Zeitpunkt nach erneutem SPRITE ON-Kommando ausführen zu lassen (mit SPRITE STOP).

Es könnte ja in unserem Spiel so sein, daß man zu Beginn beim Zusammentreffen mit irgendwelchen Wesen zerstört wird. Wurde die erste Runde geschafft, kehrt sich die Spielsituation um: Nun heißt es, mit möglichst vielen Wesen zusammenzustoßen, wobei es pro Zusammenstoß einen Bonus geben würde. Nicht auszudenken, wenn weiterhin pro Zusammenstoß eine Explosion stattfinden würde. Ganz einfach bedienen Sie sich deshalb zum Zweck des unterbundenen Unterprogrammsprungs des Kommandos SPRITE OFF. Kehrt das Spiel anschließend wieder zum Normalzustand zurück, können Sie dies Ihrem MSX-Computer mit SPRITE ON mitteilen.

Die Feststellung, daß zwei Sprites zusammenstoßen, wird nicht erst dann getroffen, wenn dargestellte Punkte der Sprites sich überlagern. Vielmehr reicht es dem Computer bereits zu dem Zeitpunkt vollkommen aus, wenn er feststellt, daß sich ein Teil ein Teil der einen Spritematrix mit dem Teil einer anderen Spritematrix auch nur in einem einzigen Punkt überlagertzusammenstößt. So kann es ohne weiteres passieren, daß sich zwei Sprites gar nicht offensichtlich auf dem Bildschirm überlagern, obgleich der MSX-Computer darüber Meldung abgibt (in diesem Fall haben sich die nicht mit Punkten besetzten Spriteelemente für uns deshalb auch nicht sichtbar übereinandergeschoben).

In einem anderen Fall ist die Spritekollisionskontrolle mit Hilfe der SPRITE ON-Anweisung gar unmöglich: Verwenden Sie mehrfarbige Sprites, bekommt Ihr MSX-Computer andauernd die Meldung einer Spritekollision. Sie müssen sich also entscheiden: Entweder Spritekolisonsüberprüfung oder mehrfarbige Sprites - beides zusammen ist nicht möglich.

Noch ein letzter Punkt zum Hinweis auf besondere Vorsicht: Erreicht den MSX-Computer die Kollisionsmitteilung, müssen Sie vorerst mit SPRITE OFF die Kollisionsüberprüfung beenden, denn beim weiteren Spielablauf kann es sonst vorkommen, daß die zwei bereits zusammengestoßenen und auch so registrierten Sprites sich immer näher kommen, bis sie sich schließlich total überlagern. In diesem Fall würde Ihr MSX-Computer andauernd die Information über neue Spritekollisionen erhalten, was den weiteren Programmablauf erheblich einschränken würde.

Ihr MSX-Computer erkennt nur, wenn Sprites zusammentreffen, aber nicht, wo und um welche Sprites es sich dabei handelt. Dies ist auch gar nicht so einfach herauszubekommen, obgleich Sie davon gerade in Spielen dringendst Mitteilung bekommen müssen (es ist zwar egal, wenn zwei Raumschiffe des gleichen Spielers zusammenstoßen, aber wehe dem, es sind zwei gegnerische Spielerobjekte). Wie ist es nun möglich, zu erfahren, welche zwei Sprites gerade zusammengestoßen sind?

Einerseits können Sie den Sprite Attribut-Tisch im VRAM daraufhin untersuchen, an welcher Koordinate sich zwei Sprites überlagern. Dies ist aber eine sehr langwierige und damit auch zeitstehlende Vorgehensweise, wie wir sie gerade bei schnellen Spielen nicht gebrauchen können. Schauen wir uns an, wie es einfacher und schneller geht, um zu erfahren, welche Sprites wo auf dem Bildschirm zusammengestoßen sind:

Während des Spielablaufs geben Sie bei der Bewegung jedes Sprites einer Variablen die Information, welches Sprite es denn war, das gerade bewegt wurde. Aufgrund dieser Information haben Sie schnellstmöglich den Kollisionsurheber ausgemacht. Das zweite Sprite (das die Kollision passiv miterlebt hat), muß sich in der gewählten Punktmatrix des aktiven Sprites aufhalten. Schauen Sie zu diesem Zweck mit VPEEK im VRAM nach und schnell ist der unschuldige Kollisionsbetroffene ausgemacht, dem dann je nach Spielablauf entweder Punkte abgezogen werden (weil er vom Gegner getroffen wurde) oder aber Punkte gutgeschrieben werden (weil er nicht der Verursacher des Zusammenstoßes war).

Wie kann man nun letztendlich feststellen, ob es sich um ein Sprite der einen oder der anderen Spielpartei handelt? Entweder man wählt die Spielparteien so, daß Spieler 1 Sprites mit ungerader Zahl besitzt (1,3,5 ...) und Spieler 2 Sprites mit gerader Zahl (2,4,6 ...) oder man gibt Spieler 1 die Sprites 1 bis 4, während Spieler 2 die Sprites 5 bis 8 sein eigen nennt. Schließlich können Sie aber auch die Sprites ungeordnet den Spielern geben (abgesehen vom dem dann natürlich steigenden Programmieraufwand zur Spritesteuerung) und bei erfolgter Spriteüberdeckung den VRAM (Sprite Attribut Table) nach der Spielerfarbe untersuchen.

Noch ein letzter Hinweis: Jeder BASIC-Befehl holt seine Informationen aus anderen Informationen des Computers, die bereits irgendwo abgespeichert vorliegen. Genau auf diese Art und Weise geschieht dies auch bei der Spritekollisionsmeldung: wenn wir das achte Register des Videoprozessors mit

```
PRINT VDP(8)
```

abfragen, erhalten wir auch dann, wenn der SPRITE OFF-Befehl gegeben wurde, die Mitteilung, wenn zwei Sprites sich überlagern: Die Ausgabe bei Überlagerung lautet bei Bit 5 in Register 8 des VDP entweder 1 (richtig) oder 0 (falsch), letzteres bei Vermeldung keiner Unfälle. Man sollte diese Abfrage aber nur dann verwenden, wenn damit eine Reaktion auf sinnvolle/sinnlose Programmunterbrechung zu erzielen bzw. zu unterbinden ist. Ansonsten würde sich Ihr Programm im Gegensatz zur bequemen Interruptabfrage durch ON SPRITE bzw. SPRITE ON/OFF allzusehr verlangsamen.

Bei der Abfrage von Register 8 müssen Sie sich allerdings darüber im klaren sein, daß hierdurch die Spritekollisionsabfrage automatisch ausgeschaltet wird (entspricht dem Kommando SPRITE OFF). Geben Sie also nach jeder Abfrage des VDP-Registers 8 - falls so gewünscht - wieder den Befehl SPRITE ON ein.

Befehle Grafik STRIG(N) ON/OFF/STOP

An- bzw. Ausstellen der Interruptsteuerung ON STRIG GOSUB

siehe hierzu auch die Programme im Anhang: STRIG(N) ON 1
STRIG(N) ON 2
STRIG(N) ON 3
STRIG(N) ON 4
STRIG 1
STRIG 2
MISTER MINER
FRETY

Eine besondere Spezialität des MSX-BASIC ist es, auch ohne Maschinensprachkenntnisse die Maschinensprache während eines BASIC-Programms durch Interruptroutinen indirekt mit einfließen lassen zu können. Die dafür zuständigen Befehle beginnen mit dem Wörtchen ON und besagen, daß nach einer einmaligen Definition diese Kontrolle vom MSX-Computer selbständig durchgeführt wird, soweit wir das Ja-Wort dazu geben (ON ERROR = wenn ein Error auftritt, ON INTERVAL = in bestimmten Zeitabständen, ON KEY = wenn eine bestimmte Funktionstaste gedrückt wurde usw.).

Der An- Aussteller dieser Interruptroutinen besagt, daß nach einem ON die selbständige Kontrolle des Computers startet, nach einem OFF die Abfrage und die darauf folgende Reaktion unterbunden wird sowie bei einem STOP die Kontrolle zwar weiterhin stattfindet, jedoch die Reaktion darauf (Verzweigen in ein Unterprogramm) unterbunden wird. Bei einem erneuten STRIG(N) ON wird der zuletzt unter STRIG(N) STOP gedrückte Joystick (falls es wirklich so geschehen ist) wahrgenommen und es wird daraufhin nachträglich in das entsprechende Unterprogramm verzweigt. Dies hat den Sinn, daß nicht aus einem laufenden Joystickunterprogramm (mit STRIG(N) ON), ohne die Zeilen abgearbeitet zu haben, wieder in ein Joystickunterprogramm (rekursiv) verzweigt wird. Erst zum Abschluß der Bearbeitung des einen Unterprogramms sollte wieder STRIG(N) ON gegeben werden, was somit die gespeicherte Information während der STRIG(N) STOP-Phase nachholt.

siehe Kurzlisting STRIG(N) ON 1 im Anhang

Der Befehl STRIG(N) ON/OFF/STOP bezieht sich auf die Interruptroutine, die die gedrückten/nicht gedrückten Joystickknöpfe und die Leertaste kontrolliert. Jedoch können durch die Interruptsteuerung nur Reaktionen gezeigt werden, wenn folgendes zutrifft:

Leertaste gedrückt: STRIG(0) ON

Joystick 1, Feuerknopf 1 gedrückt: STRIG(1) ON

Joystick 2, Feuerknopf 1 gedrückt: STRIG(2) ON

Joystick 1, Feuerknopf 2 gedrückt: STRIG(3) ON

Joystick 2, Feuerknopf 2 gedrückt: STRIG(4) ON

Die Abfrage der speziellen Joystickknöpfe für MSX (jeweils 2 Feuerknöpfe pro Joystick mit unterschiedlicher Funktion) sollte jedoch nur in dem Fall durch STRIG(3) ON bzw. STRIG(4) ON aktiviert werden, wenn Sie wahrlich stolzer Besitzer eines MSX-Joysticks sind.

Wofür ist es nun sinnvoll, die Abfrage der Joystickknöpfe an- und wieder auszuschalten?

Hantieren wir mit Joysticks (und nicht mit anderen Steuerinstrumenten, die sich ebenfalls an die Joystickbuchsen Ihres MSX-Computers anschließen lassen), gibt der Befehl STRIG(N) ON/OFF/STOP großenteils bei Spielen Sinn. Dazu drei Beispiele:

1. Sie wollen Spieler 1 in einer ersten Spielstufe nicht die Möglichkeit geben, mehrere Schüsse kurz nacheinander abzugeben (vielleicht auch aus dem Grund, weil dies den Ablauf Ihres Programms zu sehr verlangsamen würde). So warten Sie in STRIG(N) ON-Stellung ab, bis gefeuert wird, geben dann STRIG(N) OFF ein und verfolgen anschließend die Laufbahn des Pfeils. Erst wenn der Pfeil aus dem Bildschirmfenster hinausschießt, kann ein weiterer Schuß abgegeben werden (wieder STRIG(N) ON eingeben).

siehe Kurzlisting STRIG(N) ON 2 im Anhang

Um die Unterschiede von STRIG(N) ON, STRIG(N) OFF und STRIG(N) STOP kennenzulernen, verändern Sie in diesem Programm nacheinander Zeile 110 und drücken, während der Pfeil auf dem Bildschirm erscheint, immer wieder die Leertaste:

Bei STRIG(N) ON befinden sich schließlich viele Pfeile bzw. Pfeilstücke auf dem Bildschirm, da immer von neuem aus der Unterroutine verzweigt wird, um noch einen Pfeil anzuzeigen;

bei STRIG(N) OFF passiert nichts außergewöhnliches und

bei STRIG(N) STOP merkt sich Ihr MSX-Cmputer den Tasten- bzw. Joy-stickknopfdruck und führt die Funktion aus, nachdem ein erzeugtes Pfeilbild oben aus dem Bildschirm verschwunden ist.

2. Bei einem anderen Spiel darf erst dann auf den Joystickknopf gedrückt werden, wenn ein bestimmter Ton erklingt. Wieder ist uns bei diesem Ereignis der STRIG(N) ON/OFF/STOP-Befehl äußerst hilfreich:

siehe Kurzlisting STRIG(N) ON 3 im Anhang

3. Zwei Spieler dürfen abwechselnd den Feuerknopf ihres Joysticks betätigen. Mit Hilfe des STRIG(1) ON/OFF- bzw. STRIG(2) ON/OFF-Kommandos geben wir Spieler 1 und 2 abwechselnd die Möglichkeit dazu bzw. unterbinden damit gleichzeitig die Druckmöglichkeit des jeweiligen Gegenspielers.

siehe Kurzlisting STRIG(N) ON 4 im Anhang

STRIG(N) ON 1

```
                    -*-*-*-*-*-
10 SCREEN 0
20 PRINT "Bitte druecken Sie waehrend de
r"
30 PRINT "Zahlausgabe einmal die Leertas
te"
40 FOR N=1 TO 1000
50 NEXT N
60 ON STRIG GOSUB 140
70 STRIG(0) ON
80 STRIG(0) STOP
90 FOR N=1 TO 100
100 PRINT N
110 NEXT N
120 STRIG(0) ON
130 GOTO 130
140 PRINT "Leertaste gedrueckt"
```

STRIG(N) ON 2

```
                    -*-*-*-*-*-
10 SCREEN 0
20 COLOR 1,15
30 PRINT "Bitte druecken Sie zum"
40 PRINT "Schiessen die Leertaste"
50 FOR N=1 TO 1000
60 NEXT N
70 SCREEN 2
80 ON STRIG GOSUB 110
90 STRIG(0) ON
100 GOTO 100
110 STRIG(0) OFF
120 FOR N=191 TO 1 STEP -2
130 DRAW "c1 bm 128,=n; U8 N F4 G4"
140 DRAW "c15 bm 128,=n; U8 N F4 G4"
150 NEXT N
160 STRIG(0) ON
170 RETURN
```

STRIG(N) ON 3

```
                    -*-*-*-*-*-
10 SCREEN 0
20 PRINT "Bitte die Leertaste"
30 PRINT "druecken, wenn die Note C erkl
ingt"
40 PRINT
50 ON STRIG GOSUB 160
60 STRIG(0) ON
70 FOR N=1 TO 10
80 READ A$
90 IF A$="C" THEN STRIG(0) ON ELSE STRIG
(0) OFF
```

```
100 PLAY "l1"+A$
110 IF PLAY(1) THEN GOTO 110
120 NEXT N
130 PRINT "Von 3 Moeglichkeiten:";Z
140 END
150 DATA C,E,F,C,D,F,A,C,D,E
160 Z=Z+1
170 STRIG(0) OFF
180 RETURN
```

-*-*-*-*-*-

STRIG(N) ON 4

```
10 SCREEN 0
20 COLOR 1,15
30 ON STRIG GOSUB ,240,240
40 PRINT "Spieler 1 kann nun"
50 PRINT "mit Joystick 1 schiessen"
60 FOR N=1 TO 1000
70 NEXT N
80 STRIG(2) OFF
90 STRIG(1) ON
100 SCREEN 2
110 FOR N=1 TO 1000
120 NEXT N
130 SCREEN 0
140 PRINT "Spieler 2 kann nun"
150 PRINT "mit Joystick 2 schiessen"
160 FOR N=1 TO 1000
170 NEXT N
180 STRIG(2) ON
190 STRIG(1) OFF
200 SCREEN 2
210 FOR N=1 TO 1000
220 NEXT N
230 RUN
240 STRIG(1) OFF
250 STRIG(2) OFF
260 FOR M=191 TO 0 STEP-2
270 DRAW "c1 bm 128,=m; U16 N F4 G4"
280 DRAW "c15 bm 128,=m; U16 N F4 G4"
290 NEXT M
300 RETURN
```

Spritekollisionskontrolle durch Intervallsteuerung

siehe hierzu auch die Programme im Anhang: ON SPRITE 1
ON SPRITE 2

Ihr MSX-Computer ermöglicht es, innerhalb eines ablaufenden BASIC-Programms immer wieder von neuem danach Ausschau zu halten, ob etwas geschehen ist, auf das er irgendwie reagieren soll. Zu diesen Befehlen zählen u.a. ON INTERVAL und ON KEY GOSUB.

Bei der Anwendung des ON SPRITE-Befehls lassen wir den MSX-Computer nachforschen, ob irgendwann während des Programmablaufs zwei Sprites zusammenstoßen. Passiert dies, so soll er in einem Unterprogramm in eine für diesen Fall angegebene Zeile springen. Wie das folgende Programm und auch die zwei Spielprogramme in diesem Buch dies aufzeigen, findet der Befehl ON SPRITE GOSUB hauptsächlich Anwendung in Spielprogrammen: Man kann z.B. damit untersuchen, wann zwei Raketen zusammenstoßen, wie dies das folgende Programm aufzeigt:

siehe Kurzlisting ON SPRITE 1 im Anhang

Jedesmal, wenn die 2 Sprites sich in der Bildschirmmitte überlagern, ertönen kurz nacheinander 4 BEEPtöne. Die ON SPRITE-Funktion und der nachfolgende "Anstellbefehl" (SPRITE ON) in den Zeilen 30 und 40 veranlassen unseren MSX-Computer dazu, von nun an zu kontrollieren, ob sich irgendwo 2 Sprites überlagern und gegebenenfalls daraufhin in das Unterprogramm in Zeile 120 zu verzweigen.

Wir können den ON SPRITE-Befehl jederzeit wieder durch SPRITE OFF ausstellen oder die Spritekollisionskontrolle weiter durchführen, aber eine Reaktion darauf (Unterprogrammsprung) erst später durchführen lassen (erst SPRITE STOP, später SPRITE ON; das bewirkt, das entsprechende Unterprogramm wird nachträglich angesprungen - siehe auch die Beispielprogramme zum Befehl SPRITE ON).

Es ist möglich, abzufragen, welches Sprite mit welchem Sprite gerade zusammengestoßen ist: Hierzu müssen wir jede Spritebewegung in einer Variablen zwischenspeichern, wie im folgenden Programm geschehen und auch auf dem Bildschirm angezeigt:

siehe Kurzlisting ON SPRITE 2 im Anhang

Es ist entscheidend, daß die Zuweisung der Variablen Z mit den Werten 1 (für Sprite 1) bzw. 2 (für Sprite 2) vor dem eigentlichen Spritebewegungsbefehl durchgeführt wird, denn andernfalls verzweigt Ihr MSX-Computer bereits ins Unterprogramm, ohne die entsprechende Information erhalten zu haben, um welches Verursachersprite es sich handelt.

Bei diesem Programm ist es zudem ratsam (und deshalb wird es auch so durchgeführt), nach einmaligem Erreichen des Unterprogrammschleifenlaufs die ON SPRITE-Routine mit SPRITE OFF auszuschalten. Probieren Sie ruhig einmal aus, was sich sonst ereignet.

Die Sprites überlagern sich bei der nächsten kleinen Bewegung und danach noch weitere zweimal immer wieder. Erfolgt zwischenzeitlich kein SPRITE OFF-Kommando, wird Ihr MSX-Computer immer wieder ins Unterprogramm in Zeile 140 verzweigen, ohne das Unterprogramm jedes Mal zu Ende zu führen.

Der Anzeige auf dem Bildschirm können Sie entnehmen, daß immer von neuem Sprite 2 den Zusammenstoß (Zeile 110) verursacht. Hierzu soll noch eine wichtige Information gegeben werden:

Der Befehl ON SPRITE GOSUB zeigt immer dann Wirkung, wenn sich die Sprites in Ihrer Grafik irgendwie - irgendwo überlagern. Bei 2 gemalten Männchen in Form von 2 Sprites kann es somit leicht passieren, daß überhaupt kein Zusammenstoß, sondern ein "hautnaher Kontakt" bereits für die Ausführung der ON SPRITE-Routine sorgt. Um dies zu verhindern, bedarf es eines mächtigen Unterprogramms. Behelfen Sie sich vorerst damit, indem Sie Ihre Männchen nicht punktweise sondern schrittweise (mehrere Punkte nacheinander) voranschreiten lassen; dadurch bewegen sich Ihre Männchen wohl ein wenig ruckhaft, aber dafür ist die Ausführung des ON SPRITE-Kommandos auch exakt!

Übrigens:

1) Auf folgendes soll noch einmal besonders hingewiesen werden: Damit es nicht zu dem andauernden Springen ins Unterprogramm kommt, vergessen Sie nie, den SPRITE-Kollisionsbefehl durch SPRITE OFF auszuschalten.

2) Sie können das Spritekollisionsregister sowohl durch den VDP-Befehl (Bit 5 in Register 8) als auch durch das Nachschauen in den Systemvariablen ab &HF3DF abfragen. Bitte schauen Sie für nähere Informationen diesbezüglich unter der Befehlsbeschreibung zu VDP nach.

ON SPRITE 1

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 2
30 ON SPRITE GOSUB 130
40 SPRITE ON
50 SPRITE$(1)="UUUUUUUU"
60 SPRITE$(2)="UUUUUUUU"
70 FOR N=1 TO 255
80 PUT SPRITE 1,(N,96)
90 PUT SPRITE 2,(256-N,96)
100 NEXT N
110 SPRITE ON
120 GOTO 60
130 SPRITE OFF
140 BEEP
150 RETURN
```

ON SPRITE 2

```
-*-*-*-*-*-
10 COLOR 1,15
20 SCREEN 1
30 ON SPRITE GOSUB 150
40 SPRITE ON
50 SPRITE$(1)="UUUUUUUU"
60 SPRITE$(2)="UUUUUUUU"
70 FOR N=1 TO 255
80 Z=1
90 PUT SPRITE 1,(N,96)
100 Z=2
110 PUT SPRITE 2,(256-N,96)
120 NEXT N
130 SPRITE ON
140 GOTO 60
150 SPRITE OFF
160 PRINT Z;
170 CLOSE
180 SPRITE OFF
190 RETURN
```

Befehle Grafik ON STRIG GOSUB
Intervallsteuerung zum Unterprogramm bei gedrücktem Joystick

siehe hierzu auch die Programme im Anhang: ON STRIG 1
ON STRIG 2
LOCATE 9

Ebenso wie bei den bereits besprochenen Befehlen ON SPRITE GOSUB

siehe Kurzlistings ON SPRITE 1 und 2 im Anhang

und ON INTERVAL GOSUB

siehe Kurzlisting LOCATE 9 im Anhang

handelt es sich auch bei ON STRIG GOSUB um ein MSX-Kommando, das die eigentliche Programmiersprache BASIC direkt mit der Maschinensprache zusammenarbeiten läßt.

Hierbei läuft das normale BASIC-Programm nur soweit, bis die im ON-Befehl gestellte Bedingung erfüllt ist. Der Vorteil der Maschinenprogrammsteuerung mit ON ... GOSUB ist, daß wir nicht ohne Unterbrechung prüfen müssen, ob sich ein Vorgang nun ereignet hat (z.B. ob der Joystickknopf gedrückt wurde oder ob zwei Sprites zusammengestoßen sind), sondern daß dies unser MSX-Computer eigenständig tut.

Welche Bedeutung hat nun der Befehl ON STRIG GOSUB? Hiermit wird vom Computer abgefragt, ob ein Joystickfeuerknopf oder gar die Leertaste gedrückt wurde:

siehe Kurzlisting ON STRIG 1 im Anhang

Es ist in jedem Fall erforderlich, die Maschinensprachkontrolle durch ON STRIG erst einmal durch den Befehl STRIG(N) ON in Gang zu setzen, sonst tut sich überhaupt nichts! Sie können aber nach Belieben (wenn im Spiel der Feuerknopf nicht mehr gedrückt werden soll, weil man vielleicht schon selbst abgeschossen wurde) auch die automatische Abfrage des Feuerknopfs bzw. der Leertaste unterbinden, indem Sie STRIG(N) OFF eingeben. Letztendlich können Sie auch durch Eingabe des Kommandos STRIG(N) STOP veranlassen, daß sich Ihr MSX-Computer zwar das Drücken von Joystickfeuerknöpfen merkt, darauf aber erst zu einem Zeitpunkt reagiert, nachdem der Befehl STRIG(N) ON erneut eingegeben wurde.

Schauen wir uns einmal die verschiedenen Zustände an, die abgefragt werden können:

STRIG(0): Leertaste gedrückt ja/nein
STRIG(1): Feuerknopf 1 von Joystick 1 gedrückt ja/nein
STRIG(2): Feuerknopf 1 von Joystick 2 gedrückt ja/nein
STRIG(3): Feuerknopf 2 von Joystick 1 gedrückt ja/nein
STRIG(4): Feuerknopf 2 von Joystick 2 gedrückt ja/nein

In Zeile 30 könnten wir somit nun je nach Tastendruck in ein anderes Unterprogramm verzweigen, indem wir die entsprechenden Zeilennummern nacheinander durch Kommata getrennt eingeben:

siehe Kurzlisting ON STRIG 2a im Anhang

Wird z.B. die Leertaste gedrückt, verzweigt das Programm in Zeile 1000; wird Feuerknopf 2 von Joystick 2 gedrückt, wird das Programm in Zeile 5000 fortgeführt usw.

Damit auch wahrlich alle verschiedenen Joystickknöpfe kontrolliert werden, müßte anschließend in Zeile 40 bis 70 folgen:

siehe Kurzlisting ON STRIG 2b im Anhang

Wenn wir nun unser Programm noch ein klein wenig erweitern, ist die Feuerknopfabfrage perfekt und bietet auf alle nur möglichen Fragen eine Antwort:

siehe Kurzlisting ON STRIG 2c im Anhang

Nun können Sie einmal ausprobieren, inwieweit Ihre sämtlichen Joysticks auch MSX-fähig sind. Ist dies der Fall, müßten nämlich beide Feuerknöpfe (falls vorhanden) auch unterschiedliche Anzeigen auf dem Bildschirm erzeugen:

1. Joystick 1 Feuerknopf 1 gedrückt
2. Joystick 2 Feuerknopf 1 gedrückt
3. Joystick 1 Feuerknopf 2 gedrückt
4. Joystick 2 Feuerknopf 2 gedrückt

Ist es in einem Spiel angebracht, daß jeder Spieler gleich viele Knopfdruckmöglichkeiten haben soll, kann man ihn (ähnlich wie beim TV-Spiel Dalli-Klick) durch STRIG(N) OFF von weiterer Teilnahme am Spiel ausschließen.

Probieren Sie selbst einmal aus, inwieweit Sie diese Befehle (wer sprach da von Spielereien?!) in selbst programmierten Spielen nutzen können oder schauen Sie einmal in die Listings der Spiele in diesem Buch.

ON STRIG 1

```
         -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN Ø
30 ON STRIG GOSUB 6Ø
40 STRIG(Ø) ON
50 GOTO 5Ø
60 PRINT "Leertaste gedrueckt"
70 RETURN
```

ON STRIG 2a

```
         -*-*-*-*-*-
10 COLOR 1,15
20 SCREEN Ø
30 ON STRIG GOSUB 1ØØØ,2ØØØ,3ØØØ,4ØØØ,5Ø
ØØ
```

ON STRIG 2b

```
         -*-*-*-*-*-
4Ø FOR N=Ø TO 4
5Ø STRIG(N) ON
6Ø NEXT N
```

ON STRIG 2c

```
         -*-*-*-*-*-
999 GOTO 999
1ØØØ PRINT "Leertaste gedrueckt"
1Ø1Ø RETURN
2ØØØ PRINT "Joystick 1, Feuerknopf 1 ged
rueckt"
2Ø1Ø RETURN
3ØØØ PRINT "Joystick 2, Feuerknopf 1 ged
rueckt"
3Ø1Ø RETURN
4ØØØ PRINT "Joystick 1, Feuerknopf 2 ged
rueckt"
4Ø1Ø RETURN
5ØØØ PRINT "Joystick 2, Feuerknopf 2 ged
rueckt"
5Ø1Ø RETURN
```

Kurzbeschreibung zu Programm 1

Der Video-RAM wird gelöscht, wenn Sie einen SCREEN-Befehl zwecks Bildschirmänderung eingeben z.B. zum Sprung von der Textseite (SCREEN 0 bzw. SCREEN 1) auf eine der Grafikseiten (SCREEN 2 bzw. SCREEN 3).

Der Sprite Generator Table im Video-RAM wird gelöscht, wenn Sie den zweiten Parameter im SCREEN-Befehl (z.B. SCREEN ,1) ändern.

Letzteres hat zur Folge, daß Sie bei Änderung der Spritegröße jedesmal alle bereits vorher verwendeten Spritemuster neu definieren bzw. neu aus DATA-Zeilen einlesen müssen.

Mit dem VDP-Befehl haben Sie die Möglichkeit, die Spritegröße im Programmablauf zu ändern. Dadurch werden die bereits zuvor erstellten Spritemuster im VRAM nicht gelöscht.

Programm 1 zeigt die Anwendung des VDP-Befehls in diesem Sinne recht eindrucksvoll: Hier werden Sprites in der 8*8-Matrix erst in normaler Größe (ein Spritepunkt entspricht einem Bildschirmpunkt = SCREEN ,0) und anschließend in der Vergrößerungsstufe abgebildet (ein Spritepunkt entspricht 4 Bildschirmpunkten = SCREEN ,1). Dabei wird das Spritemuster trotz Größenänderung in seinem Aussehen nicht verändert d.h. das Laden des Spritemusters findet nur zu Beginn des Programms statt.

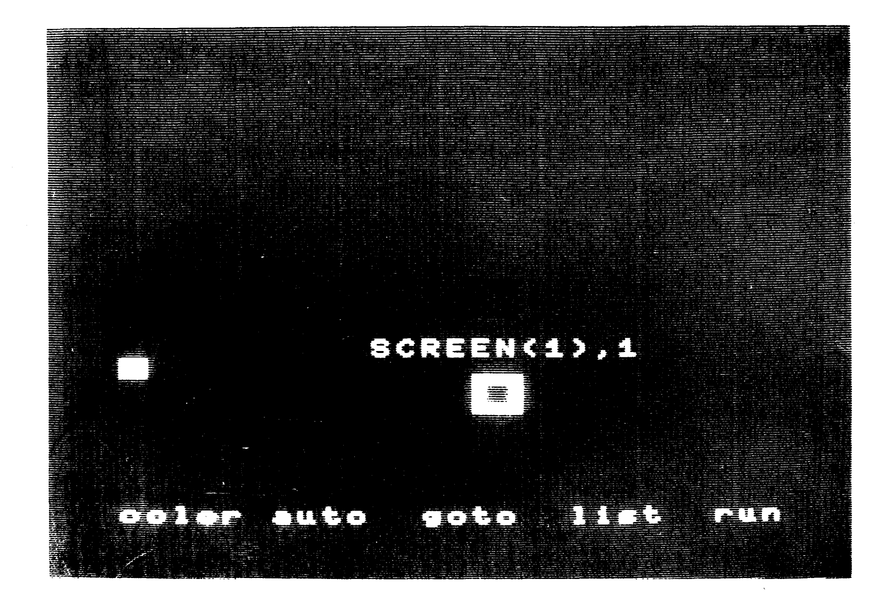
Sicherlich ist Ihnen durch dieses Programm bei Schwierigkeiten mit der Spritedarstellung etwas weitergeholfen. Idealer wäre es allerdings, wenn sämtliche Spritegrößen auf dem Bildschirm gleichzeitig dargestellt werden könnten.

Programm 1   Teil 1

```
10 REM Programm 1
20 REM MSX Grafik und Sound
30 REM Copyright 1985 DATA BECKER &
                        Rainer Lüers
40 REM 8*8-Sprites normal und
       vergrößert, abwechselnd
       dargestellt ohne VRAM-Löschung
50 SCREEN 1
60 COLOR 15,4
70 REM Sprite 1 = Schachbrettmuster
80 DATA 01010101
90 DATA 10101010
100 DATA 01010101
110 DATA 10101010
120 DATA 01010101
130 DATA 10101010
140 DATA 01010101
150 DATA 10101010
160 REM Sprite 2 = Rahmen
170 DATA 11111111
180 DATA 11111111
190 DATA 11000011
200 DATA 11000011
210 DATA 11000011
220 DATA 11000011
230 DATA 11111111
240 DATA 11111111
250 REM Einlesen der Sprites 0 und 1
260 RESTORE 80
270 FOR M=0 TO 1
280 B$=""
290 FOR N=0 TO 7
300 READ A$
310 REM Umwandlung Binärdarstellung
                      Charakterzeichen
320 B$=B$+CHR$(VAL("&b"+A$))
330 NEXT N
```

Programm 1   Teil 2

```
340 SPRITE$(M)=B$
350 NEXT M
360 FOR A=1 TO 100
370 NEXT A
380 REM Abbildung der Sprites Ø und 1
390 PUT SPRITE Ø,(128,86),1,Ø
400 PUT SPRITE 1,(128,126),15,1
410 REM Abfrage von Bit Ø
        in VDP-Register 1
420 B=VDP(1) AND 1
430 REM Wenn Bit Ø = 1,
        dann ist nun Bit Ø = Ø
440 IF B=1 THEN VDP(1)=VDP(1) AND &B1111
1110:LOCATE 10,14:PRINT "SCREEN(1),Ø":GO
TO 360
450 REM Wenn Bit Ø = Ø,
        dann ist nun Bit Ø = 1
460 IF B=Ø THEN VDP(1)=VDP(1) OR &BØØØØØ
ØØ1:LOCATE 10,14:PRINT "SCREEN(1),1":GOT
O 360
```



Kurzbeschreibung zu Programm 2

Sie haben auf SCREEN 0 die Möglichkeit, bis zu 960 Zeichen (24 Zeilen zu je 40 Zeichen) zur gleichen Zeit auf dem Bildschirm (und damit verbunden selbstverständlich auch im VRAM) darzustellen bzw. abzulegen.

Der VRAM (über 16000 Speicherplätze) wird bei SCREEN 0 durch diese 960 Zeichen und das Aussehen der Zeichen (2048 Speicherplätze) nur zum Teil genutzt. Die frei gebliebenen Speicherplätze können nun z.B. zur Variablenabspeicherung über VPEEK und VPOKE oder durch den Befehl SPRITE$ genutzt werden, sicherlich kein erstrebenswertes Ziel.

In Programm 2 wird der VRAM bei Anwahl von SCREEN 0 dadurch besser genutzt, indem hier gleichzeitig mehrere Textbildschirme den Speicher ausmachen. Mit einem VDP-Befehl haben Sie die Möglichkeit, wie in diesem Programm auch gezeigt, bis zu 13 verschiedene Textbildschirme im VRAM nicht nur abzulegen, sondern sie auch durch einen schnellen Abruf wieder auf den Bildschirm zu bringen.

Wofür wäre das sinnvoll? Sie könnten z.B. in diesen 13 verschiedenen Bildschirmen Hilfsinformationen oder oft zu verwendende Formeln abspeichern. Ihrer Phantasie sind bei diesem Ultrakurzwechsel der Bildschirme keine Grenzen gesetzt (z.B. könnten Sie auf diesen 13 Bildschirmen einen kleinen Zeichentrickfilm entwickeln).

Programm 2 Teil 1

```
10 REM Programm 2
20 REM MSX Grafik und Sound
30 REM Copyright 1985 DATA BECKER &
                      Rainer Lüers
40 REM Wir arbeiten in SCREEN 0 mit bis
       zu 13 eigenständigen Bildschirmen
       gleichzeitig
50 SCREEN 0
60 WIDTH 39
70 KEY OFF
80 REM Bei Programmunterbrechung immer
       auf den Eingabebildschirm
       zurückkehren
90 ON STOP GOSUB 430
100 STOP ON
110 M1=200
120 REM Die Bildschirme 1 bis 12 werden
        mit einem Kurzhinweis belegt
130 FOR N=1 TO 12
140 PRINT "Initialisierung von Bildschir
m";N
150 A$="Das ist Bildschirm"+STR$(N)+STRI
NG$(10,32)
160 REM POKEn des Hinweissatzes (A$)
        an die entsprechende Anfangs-
        position der 12 Bildschirme
170 FOR M=1 TO LEN(A$)
180 VPOKE (3+N)*1024+M,ASC(MID$(A$,M,1))
190 NEXT M
200 NEXT N
210 CLS
220 PRINT "Das ist Bildschirm 0"
230 VDP(2)=0
240 REM Zeitschleife anspringen,
        falls gewünscht
250 GOSUB 390
260 REM Blättern in den Bildschirmen
```

Programm 2   Teil 2
==========

```
        1 bis 12
270 FOR N=4 TO 15
280 VDP(2)=N
290 REM Zeitschleife anspringen,
        falls gewünscht
300 GOSUB 390
310 NEXT N
320 VDP(2)=0
330 PRINT
340 B$=""
350 INPUT "Bitte Geschwindigkeit eingebe
n          (1000 bis 1) ";B$
360 M1=VAL(B$)
370 GOTO 230
380 REM Zeitschleife
390 FOR M=1 TO M1
400 NEXT M
410 RETURN
420 REM Rückkehr auf den Eingabebild-
        schirm und Programmende bei
        Drücken der Tasten CTRL STOP
430 VDP(2)=0
440 END
```

```
Initialisierung von Bildschirm 1
Initialisierung von Bildschirm 2
Initialisierung von Bildschirm 3
Initialisierung von Bildschirm 4
Initialisierung von Bildschirm 5
Initialisierung von Bildschirm 6
Initialisierung von Bildschirm 7
Initialisierung von Bildschirm 8
Initialisierung von Bildschirm 9
Initialisierung von Bildschirm 10
Initialisierung von Bildschirm 11
Initialisierung von Bildschirm 12
```

Kurzbeschreibung zu Programm 3

Wenn Sie mit Ihrem MSX-Computer auf den Textbildschirmen SCREEN 0 bzw. SCREEN 1 arbeiten, haben Sie die Möglichkeit, bis zu 256 verschiedene Zeichen gleichzeitig darzustellen.

Wir können uns einen Teil dieser Zeichen (die Zeichen, die auf dem Bildschirm sichtbar darstellbar sind) mit folgendem Programm anzeigen lassen:

```
10 FOR N=32 TO 255
20 PRINT CHR$(N);
30 NEXT N
```

Verändern Sie ein Zeichen in seinem Aussehen, verändern sich gleichzeitig sämtliche Zeichen mit gleichem Zeichencode auf dem Bildschirm (siehe in diesem Buch unter CHR$ und ASC).

Leider können auch mit Programm 3 nicht gleichzeitig mehr als 256 verschiedene Zeichen dargestellt werden. Dafür bietet Ihnen dieses Programm aber die Möglichkeit, nicht nur zwei unterschiedliche Zeichensätze im VRAM abzulegen, sondern dieselbigen auch in Bruchteilen von Sekunden auszutauschen. Hierzu leistet uns wieder einmal der VDP-Befehl wertvolle Hilfestellung.

Um das Programm kurz zu halten, habe ich als zweiten Zeichensatz den bereits bekannten Standardzeichensatz übernommen, allerdings die Großbuchstaben zu Inverszeichen umgewandelt.

Programm 3 Teil 1

```
10 REM Programm 3
20 REM MSX Grafik und Sound
30 REM Copyright 1985 DATA BECKER &
                          Rainer Lüers
40 REM Programm zur Verwendung von 2
       verschiedenen Zeichensätzen,
       die sich mit VDP(4) leicht und
       schnell umschalten lassen
50 SCREEN 0
60 WIDTH 40
70 PRINT "Ein zweiter Zeichensatz mit in
versen    Grossbuchstaben wird abgespeic
hert"
80 PRINT
90 REM Die Charakterzeichen 0 bis 64
       werden mit gleichem Aussehen
       für den 2.Zeichensatz über-
       nommen
100 FOR N=0 TO 64
110 PRINT N;
120 FOR M=1 TO 8
130 GOSUB 660
140 NEXT M
150 NEXT N
160 REM Die Charakterzeichen 65 bis 90
        (Großbuchstaben) werden für
        den zweiten Zeichensatz umge-
        rechnet und erhalten dadurch
        ein inverses Aussehen
170 FOR N=65 TO 90
180 PRINT N;
190 FOR M=1 TO 8
200 A$=BIN$(VPEEK(2047+(N*8)+M))
210 A$=STRING$(8-LEN(A$),"0")+A$
220 REM Binäraufbau umdrehen (d.h. wo
        eine 0 ist, kommt eine 1 und
        umgekehrt)
```

Programm 3   Teil 2
==========

```
230 FOR L=1 TO 8
240 IF MID$(A$,L,1)="0" THEN MID$(A$,L,1
)="1" ELSE MID$(A$,L,1)="0"
250 NEXT L
260 VPOKE 4095+(N*8)+M,VAL("&B"+A$)
270 NEXT M
280 NEXT N
290 REM Die Charkterzeichen 91 bis 254
        werden mit gleichem Aussehen
        für den zweiten Zeichensatz
        übernommen
300 FOR N=91 TO 254
310 PRINT N;
320 FOR M=1 TO 8
330 GOSUB 660
340 NEXT M
350 NEXT N
360 REM Das Cursorzeichen wird zum
        Schachbrettmuster umgewandelt
370 FOR M=1 TO 8 STEP 2
380 VPOKE 4095+(255*8)+M,85
390 VPOKE 4095+(255*8)+M+1,170
400 NEXT M
410 CLS
420 PRINT "Zur Zeit sieht Ihr Zeichensat
z so aus:"
430 PRINT
440 REM Ausdruck des auf dem Bild-
        schirm sichtbaren Zeichen-
        satzes
450 FOR N=32 TO 255
460 PRINT CHR$(N);
470 NEXT N
480 PRINT
490 PRINT
500 GOSUB 690
510 PRINT "Wir schalten jetzt auf den an
```

Programm 3   Teil 3
==========

```
deren       Zeichensatz (GROSSBUCHSTABEN
INVERS)     mit VDP(4)=2 um"
520 PRINT
530 GOSUB 690
540 REM Umschaltung auf Zeichensatz 2
550 VDP(4)=2
560 PRINT "Und nun 1000 schnelle Wechsel
            zwischen den 2 Zeichensaetzen
"
570 GOSUB 690
580 FOR N=1 TO 1000
590 REM Zeichensatz 1 Aufruf
600 VDP(4)=1
610 REM Zeichensatz 2 Aufruf
620 VDP(4)=2
630 NEXT N
640 END
650 REM Kopieren des Zeichensatzes
660 VPOKE 4095+(N*8)+M,VPEEK(2047+(N*8)+
M)
670 RETURN
680 REM Warteschleife
690 FOR N=1 TO 1000
700 NEXT N
710 RETURN
```

```
Ein zweiter Zeichensatz mit inversen
Grossbuchstaben wird abgespeichert

 0   1   2   3   4   5   6   7   8   9   10   11
 12   13   14   15   16   17   18   19   20   21
 22   23   24   25   26   27   28   29   30   31
 32   33   34   35   36   37   38   39   40   41
 42   43   44   45   46   47   48   49   50   51
 52   53   54   55   56   57   58   59   60   61
 62   63   64   65   66   67   68   69   70   71
 72   73   74   75   76   77   78   79   80   81
 82   83   84   85   86   87   88   89   90   91
 92   93   94   95   96   97   98   99   100
 101   102   103   104   105   106   107   108
 109   110   111   112   113   114   115   116
 117   118   119   120   121   122   123   124
 125   126   127   128   129   130   131   132
 133   134   135   136   137   138   139   140
 141   142   143   144   145   146   147   148
 149   150   151   152   153   154   155   156
 157   158   159   160   161   162   163   164
 165   166   167   168   169   170   171   172
 173   174   175   176   177   178
```

```
Zur Zeit sieht Ihr Zeichensatz so aus:

 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~Çüéâäàåçêëèïîì ÄÅÉæÆôöòûùÿÖ
Ü¢£¥₧ƒáíóúñÑªº¿⌐¬½¼¡«»

Wir schalten jetzt auf den anderen
Zeichensatz (GROSSBUCHSTABEN INVERS)
mit VDP(4)=2 um

Und nun 1000 schnelle Wechsel
zwischen den 2 Zeichensaetzen
Ok
```

Kurzbeschreibung zu Programm 4

Sie haben die Möglichkeit, entweder 8*8-Sprites oder 16*16-Sprites jeweils in normaler und doppelter Größe auf dem Bildschirm darzustellen.

Sie können im VRAM-Speicher bis zu 256 verschiedene Sprites in der Größe 8*8 oder 64 verschiedene Sprites in der Größe 16*16 ablegen.

Von diesen 256 bzw. 64 Sprites können Sie bis zu 32 gleichzeitig auf dem Bildschirm darstellen, vorausgesetzt Sie beachten folgende Regel: es können in jeder horizontalen Bildschirmlinie nur bis zu 4 Sprites gleichzeitig dargestellt werden. Ist dies nicht der Fall, so kommt es zur suksessiven oder sofortigen Löschung des Sprites mit der höchsten Spritenummer auf dem aktuellen Bildschirm.

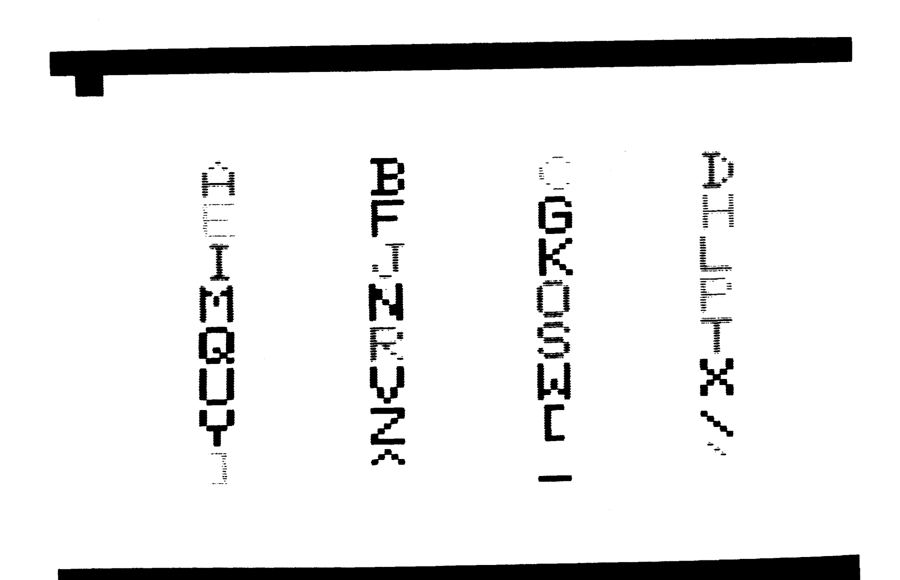
Programm 4 zeigt Ihnen nicht nur, wie Sie 32 Sprites gleichzeitig auf dem Bildschirm ohne horizontale Bildüberlappung darstellen können. Vielmehr können Sie durch das Programmlisting auch erfahren, wie mit dem VDP-Kommando sämtliche Sprites mit dem normalen Zeichensatz geladen werden können.

Programm 4 Teil 1

```
10 REM Programm 4
20 REM MSX Grafik und Sound
30 REM Copyright 1985 DATA BECKER &
                      Rainer Lüers
40 REM Belegen der 32 Sprites mit den
       großen Buchstaben (aus dem
       normalen Charakterzeichensatz
       entnommen)
50 REM 8*8-Spites in vergrößerter
       Darstellung
60 SCREEN 1,1
70 KEY OFF
80 COLOR 1,15
90 REM Das Aussehen der Sprites (VDP(6))
       ergibt sich aus dem Aussehen der
       Charakter (VDP(4))
100 VDP(6)=VDP(4)
110 M1=0
120 REM 32 Sprites auf dem Bildschirm
        übersichtlich darstellen
130 FOR N=32 TO 152 STEP 16
140 FOR M=50 TO 200 STEP 50
150 REM Nur die Sprites mit Großbch-
        staben werden abgebildet (A=65);
        jedem Sprite wird eine Farbe
        zugeordnet
160 PUT SPRITE M1,(M,N),INT(RND(1)*14)+1
,M1+65
170 M1=M1+1
180 NEXT M
190 NEXT N
200 REM Pixelweises Scrolling der
        32 Sprites gleichzeitig
210 FOR L=1 TO 255
220 M1=0
230 FOR N=32 TO 152 STEP 16
240 FOR M=50 TO 200 STEP 50
```

Programm 4 Teil 2

```
250 PUT SPRITE M1,(M+L,N),,M1+65
260 M1=M1+1
270 NEXT M
280 NEXT N
290 NEXT L
```



Kurzbeschreibung zu Programm 5

Sie haben die Möglichkeit, entweder 8*8-Sprites oder 16*16-Sprites jeweils in normaler und doppelter Größe auf dem Bildschirm darzustellen.

Sie können im VRAM-Speicher bis zu 256 verschiedene Sprites in der Größe 8*8 oder 64 verschiedene Sprites in der Größe 16*16 ablegen.

Von diesen 256 bzw. 64 Sprites können Sie bis zu 32 gleichzeitig auf dem Bildschirm darstellen, vorausgesetzt Sie beachten folgende Regel: es können in jeder horizontalen Bildschirmlinie nur bis zu 4 Sprites gleichzeitig dargestellt werden. Ist dies nicht der Fall, so kommt es zur suksessiven oder sofortigen Löschung des Sprites mit der höchsten Spritenummer auf dem aktuellen Bildschirm.

Programm 5 demonstriert Ihnen in mehreren Beispielen, wie diese Bildüberlappung suksessive abläuft d.h. welches Sprite von der Bildüberlappung betroffen ist, wenn sich mehr als 4 Sprites gleichzeitig in einer horizontalen Linie begegnen.

Programm 5   Teil 1

```
10 REM Programm 5
20 REM MSX Grafik und Sound
30 REM Copyright 1985 DATA BECKER &
                        Rainer Lüers
40 REM Demonstrationsprogramm, um zu
       zeigen, daß nur bis zu 4 Sprites
       auf einer Horizontalen gleich-
       zeitig erscheinen können; das
       Sprite mit der größten Zahl wird
       suksessive gelöscht
50 REM 8*8-Sprites in vergrößerter
       Darstellung
60 SCREEN 1,1
70 KEY OFF
80 COLOR 1,15
90 WIDTH 29
100 REM Das Aussehen der Sprites (VDP(6)
)       ergibt sich aus dem Aussehen der
        Charakter (VDP(4))
110 VDP(6)=VDP(4)
120 REM 3 Zustände werden überprüft
130 FOR L=1 TO 3
140 READ N0,N1,N2,N3,N4
150 REM Anzeige der feststehenden Sprite
s
160 PUT SPRITE N0,(25,96),,65
170 PUT SPRITE N1,(75,96),,66
180 PUT SPRITE N3,(150,96),,68
190 PUT SPRITE N4,(200,96),,69
200 LOCATE 1,10
210 REM Anzeige der Spritezahlen
220 PRINT N0;TAB(7);N1;TAB(11);N2;TAB(16
);N3;TAB(22);N4
230 REM Das 5.Sprite stößt hinzu
240 FOR N=70 TO 120
250 PUT SPRITE N2,(112,N),,67
260 REM Ausgabe des Inhalts von VDP(8)
```

Programm 5   Teil 2
==========

```
270 A$=BIN$(VDP(8))
280 LOCATE Ø,1
290 PRINT "VDP(8)=";A$
300 PRINT "mehr als 4 Sprites (1=ja):";M
ID$(A$,2,1)
310 PRINT "ueberdecktes Sprite:";:IF MID
$(A$,4,5)="11111" THEN PRINT "keins" ELS
E PRINT VAL("&b"+MID$(A$,4,5));"    "
320 REM Warteschleife
330 FOR M=1 TO 200
340 NEXT M
350 NEXT N
360 NEXT L
370 LOCATE Ø,Ø
380 END
390 REM DATAS von 3 Zuständen
400 DATA Ø,1,4,2,3
410 DATA Ø,1,2,3,4
420 DATA 4,3,2,1,Ø
```

```
VDP(8)=11000100
mehr als 4 Sprites (1=ja):1
ueberdecktes Sprite: 4
```

Kurzbeschreibung zu Programm 6

Es werden Ihnen in diesem Buch verschiedenste Möglichkeiten gezeigt, wie Sie die Zeichen auf dem Bildschirm in ihrem Aussehen verändern können.

Das Problem nach einer gelungenen Zeichensatzänderung ist leider, daß Ihr Computer nach dem Ausschalten oder auch bereits nach einem SCREEN-Wechsel sämtliche vorgenommenen Änderungen wieder vergißt und zum Standardformat zurückkehrt.

Für Sie wird es deshalb schnell Probleme geben, wenn Sie sich geänderte Zeichensätze im Aussehen merken wollen.

Programm 6 bietet Ihnen nun die Möglichkeit, jeden beliebigen Zeichensatz auf Kassette abzuspeichern, vorausgesetzt, die Änderung wurde auf SCREEN 1 vorgenommen.

Nicht nur das: Die Abspeicherung auf Kassette erfolgt in Form von Zeilennummern und entsprechenden DATA-Zeilen. So können Sie den geänderten Zeichensatz zu einem späteren Zeitpunkt mit dem Kommando MERGE in ein anderes Programm direkt wieder einbinden.

```
Zeichensatz abspeichern
-----------------------
Anfangswert ? 33
Endwert ? 200

So sehen die von Ihnen zum
Abspeichern ausgewaehlten
Zeichen aus:

!"#$%&'()*+,-./0123456789:;<=
>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ
[\]^_`abcdefghijklmnopqrstuvw
xyz{|}Çüéâäàåçêëèïîì ÄÅÉæÆôöòû
ùÿÖÜ¢£¥₧ƒáíóúñÑªº¿⌐¬½¼¡«»Ãã Ĩĩ
ÕõŨũĲĳ¾∽◊‰¶§

Ist das so richtig (/N) ?

Zeichensatzname ? MSX

Recorder auf Aufnahme

Fertig ( / ) ? █
```

Programm 6   Teil 1

```
10 REM Programm 6
20 REM MSX Grafik und Sound
30 REM Copyright 1985 DATA BECKER &
                       Rainer Lüers
40 REM Progrmm, um Zeichensätze aus dem
       Speicher (SCREEN 1) auf Kassette
       abzulegen mit der Möglichkeit,
       sie später in Form von DATA-
       Zeilen + Laderoutine in ein be-
       reits bestehendes Programm zu
       MERGEn
50 CLS
60 KEY OFF
70 REM SCREEN 1 muß angewählt sein
80 WIDTH 29
90 PRINT "Zeichensatz abspeichern"
100 PRINT "-----------------------"
110 REM Eingrenzen des Zeichenatzes
120 INPUT "Anfangswert ";M1
130 INPUT "Endwert ";M2
140 IF M1>M2 OR M2>255 OR M1<0 THEN RUN
150 PRINT
160 REM Anzeige der gewählten Zeichen
170 PRINT "So sehen die von Ihnen zum
Abspeichern ausgewaehlten    Zeichen aus
:"
180 PRINT
190 FOR N=M1 TO M2
200 PRINT CHR$(N);
210 NEXT N
220 PRINT
230 PRINT
240 INPUT "Ist das so richtig ( /N) ";F$
250 IF F$<>"" THEN RUN
260 F$=""
270 PRINT
280 INPUT "Zeichensatzname ";F$
```

Programm 6   Teil 2
==========

```
290 PRINT
300 PRINT "Recorder auf Aufnahme"
310 PRINT
320 INPUT "Fertig ( / ) ";F1$
330 F$="cas:"+F$
340 OPEN F$ FOR OUTPUT AS #1
350 REM Setzen der Anfangszeile auf 1000
360 Z=1000
370 REM Laderoutine wird erstellt und
        abgespeichert
380 A$=STR$(Z)+" FOR N ="+STR$(M1)+" TO"
+STR$(M2)
390 GOSUB 850
400 A$=STR$(Z)+" FOR M=0 TO 7"
410 GOSUB 850
420 A$=STR$(Z)+" READ A$"
430 GOSUB 850
440 A$=STR$(Z)+" VPOKE N*8+M"+CHR$(44)+"
VAL("+CHR$(34)+"&B"+CHR$(34)+"+A$)"
450 GOSUB 850
460 A$=STR$(Z)+" NEXT M"
470 GOSUB 850
480 A$=STR$(Z)+" NEXT N"
490 PRINT #1,A$
500 REM Abspeichern des Zeichenaussehens
        in Binärform
510 FOR N=M1 TO M2
520 REM Anzeige der z.Z. abgespeicherten
        Zeichen und Programmzeilen
530 LOCATE 16,CSRLIN-1
540 PRINT "Zeichen";N
550 LOCATE 16,CSRLIN
560 PRINT "Zeile ";
570 PRINT USING "####";Z+10;
580 Z=Z+10
590 REM Erstellen einer Programmzeile
        mit Zeilennummer
```

Programm 6   Teil 3
==========

```
600 A$=STR$(Z)+" REM  Zeichen "+STR$(N)
610 PRINT #1,A$
620 FOR M=0 TO 7
630 A$=""
640 Z=Z+10
650 A$=STR$(Z)+" DATA "
660 REM Lesens des Zeichens aus dem
        Speicher und Umwandlung in
        Binärdarstellung
670 A1$=BIN$(VPEEK(N*8+M))
680 A1$=STRING$(8-LEN(A1$),"0")+A1$
690 A$=A$+A1$
700 REM Abspeicherung der vollständigen
        Zeile Z
710 PRINT #1,A$
720 NEXT M
730 NEXT N
740 CLOSE
750 END
760 REM Testroutine, um die abgespei-
        cherte Datei auf Kassette an-
        zuschauen
770 CLS
780 OPEN "cas:"FOR INPUT AS #1
790 LINE INPUT #1,A$
800 PRINT A$
810 REM Wenn keine Datas mehr auf Kas-
        sette, dann schließe die Datei
820 IF EOF(1)=-1 THEN CLOSE:END
830 GOTO 790
840 REM Abspeichern der Programmzeilen
        und Heraufsetzen der Zeilen-
        nummer Z um 10
850 PRINT #1,A$
860 Z=Z+10
870 RETURN
```

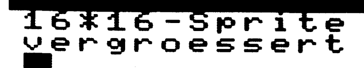
Kurzbeschreibung zu Programm 7

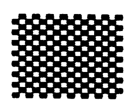
Sie haben die Möglichkeit, entweder 8*8-Sprites oder 16*16-Sprites jeweils in normaler und doppelter Größe auf dem Bildschirm darzustellen.

Sie können im VRAM-Speicher bis zu 256 verschiedene Sprites in der Größe 8*8 oder 64 verschiedene Sprites in der Größe 16*16 ablegen.

Von diesen 256 bzw. 64 Sprites können Sie bis zu 32 gleichzeitig auf dem Bildschirm darstellen, vorausgesetzt Sie beachten folgende Regel: es können in jeder horizontalen Bildschirmlinie nur bis zu 4 Sprites gleichzeitig dargestellt werden. Ist dies nicht der Fall, so kommt es zur suksessiven oder sofortigen Löschung des Sprites mit der höchsten Spritenummer auf dem aktuellen Bildschirm.

Mit Programm 7 haben Sie die Möglichkeit, nach einmaliger Definition des Spriteaussehens zwischen sämtlichen oben angeführten Spritegrößen hin- und her zu wechseln. Behilflich ist Ihnen bei dieser Vorgehensweise wieder einmal der VDP-Befehl: Im Gegensatz zum SCREEN-Befehl beiben somit bei richtigem Einsatz des VDP-Befehls sämtliche Informationen im VRAM erhalten.





Programm 7   Teil 1

```
10 REM Programm 7
20 REM MSX Grafik und Sound
30 REM Copyright 1985 DATA BECKER &
                      Rainer Lüers
40 REM Umschaltung zwischen 8*8- und
       16*16-Spritedarstellung, jeweils
       normale Größe und doppelte Größe
       ohne SCREEN-Befehl und ohne Neu-
       definition bei Umschaltung
50 SCREEN 1
60 KEY OFF
70 COLOR 1,15
80 REM Einlesen des Srites
       (16*16 Punkte)
90 FOR N=1 TO 16
100 READ A$
110 REM Erst wird die linke Seite einge-
        lesen (Sprite 1 und 2 bei 8*8)
120 A1$=A1$+CHR$(VAL("&B"+LEFT$(A$,8)))
130 NEXT N
140 RESTORE
150 FOR N=1 TO 16
160 READ A$
170 REM Hier wird die rechte Seite der
        Spritedarstellung gelsesen
        (Sprite 3 und 4 bei 8*8)
180 A1$=A1$+CHR$(VAL("&B"+RIGHT$(A$,8)))
190 NEXT N
200 REM Zuweisung des Spritemusters an
        den SPRITE PATTERN TABLE
210 FOR N=14336 TO 14367
220 VPOKE N,ASC(MID$(A1$,N-14335,1))
230 NEXT N
240 REM großes Schachbrett-Sprite
250 DATA 0101010101010101
260 DATA 1010101010101010
270 DATA 0101010101010101
```

Programm 7 Teil 2
==========

```
280 DATA 1010101010101010
290 DATA 0101010101010101
300 DATA 1010101010101010
310 DATA 0101010101010101
320 DATA 1010101010101010
330 DATA 0101010101010101
340 DATA 1010101010101010
350 DATA 0101010101010101
360 DATA 1010101010101010
370 DATA 0101010101010101
380 DATA 1010101010101010
390 DATA 0101010101010101
400 DATA 1010101010101010
410 CLS
420 LOCATE 0,0
430 PRINT "8*8-Sprite"
440 PRINT "nicht vergroessert"
450 REM Bit 0 und Bit 1 von Register 1
        werden auf 0 gesetzt d.h.
        8*8-Sprite normale Größe
460 VDP(1)=VDP(1) AND &HFC
470 GOSUB 740
480 CLS
490 LOCATE 0,0
500 PRINT "8*8-Sprite"
510 PRINT "vergroessert"
520 REM Bit 0 von Register 1
        wird auf 1 gesetzt d.h.
        8*8-Sprite doppelte Größe
530 VDP(1)=VDP(1) OR 1
540 GOSUB 740
550 CLS
560 REM Bit 0 und Bit 1 von Register 1
        werden auf 0 gesetzt d.h.
        8*8-Sprite normale Größe
570 VDP(1)=VDP(1) AND &HFC
580 LOCATE 0,0
```

Programm 7   Teil 3
==========

```
590 PRINT "16*16-Sprite"
600 PRINT "nicht vergroessert"
610 REM Bit 1 von Register 1
        wird auf 1 gesetzt d.h.
        16*16-Sprite normale Größe
620 VDP(1)=VDP(1) OR 2
630 GOSUB 740
640 CLS
650 LOCATE 0,0
660 PRINT "16*16-Sprite"
670 PRINT "vergroessert"
680 REM Bit 0 von Register 1
        wird auf 1 gesetzt d.h.
        16*16-Sprite doppelte Größe
690 VDP(1)=VDP(1) OR 1
700 GOSUB 740
710 REM Neustart der Demonstration,
        ohne das Spritemuster neu
        laden zu müssen
720 GOTO 410
730 REM Unterprogramm zur
        Spritebewegung diagonal
740 FOR N=1 TO 191
750 PUT SPRITE 0,(N,N),1,0
760 FOR M=1 TO 10
770 NEXT M
780 NEXT N
790 RETURN
```

Kurzbeschreibung zu Programm 8

Ihr MSX-Computer bietet durch den eingebauten Tonchip nicht nur die Möglichkeit, einzelne Töne zu erzeugen, sondern sogar dieselbigen durch drei Tonkanäle gleichzeitig zu spielen.

Sie haben die Möglichkeit, diese Klänge durch Direktansteuerung der einzelnen Register im Tonchip oder durch den MSX-BASIC-Befehl PLAY zu erzeugen.

Jedoch müssen Sie sich als kundiger Klavierspieler an den Notennamen (bei PLAY) bzw. diversen Zahleingaben (SOUND) orientieren, um Musik zu erzeugen - eine Klaviatur ist leider nicht eingebaut.

Um diesen Mißstand zu beseitigen, habe ich Programm 8 geschrieben: Mit diesem Programm können Sie Ihre Computertastatur so umfunktionieren, daß einzelne Töne direkt auf Tastendruck erzeugt werden. Welche Tasten gedrückt werden können wird Ihnen in Kombination zu der entsprechenden Taste am Bildschirm angezeigt.

Außerdem wird Ihnen parallel zum gespielten Ton die Note ausgegeben, so daß Sie in der Kombination von 'Taste drücken' und 'Ton hören' auch noch Noten lernen können.

Schließlich können Sie mit Funktionstaste 1 bis 5 Dreiklänge abrufen, die Sie vorher direkt mit Ihrer Tastatur eingegeben haben (die Eingabe erfolgt durch vorheriges Löschen des Akkordfeldes durch SHIFT mit der entsprechenden Funktionstaste z.B. Akkord 1 (F1) löschen und neu eingeben durch F6).

Programm 8 Teil 1

```
10 REM Programm 8
20 REM MSX Grafik und Sound
30 REM Copyright 1985 DATA BECKER &
                      Rainer Lüers
40 REM Musik erzeugen auf eineinhalb Ok-
       taven mit Klaviatur, Notenanzeige
       u.Akkordabruf bzw.Akkorderzeuger
50 REM Ausführung, was die Akkorderzeu-
       gung betrifft, nach Funktionsta-
       stendruck
60 COLOR 15,4,4
70 ON KEY GOSUB 1280,1300,1320,1340,1360
,1390,1420,1450,1480,1510
80 FOR N=1 TO 10
90 KEY(N) ON
100 KEY N,""
110 NEXT N
120 REM Ein vergrößerter 8*8-Spritepfeil
        zeigt den letzten Tastendruck an
130 SCREEN 2,1
140 REM Zeichnen der weißen
        Klaviertasten
150 FOR N=24 TO 224 STEP 16
160 LINE (N,80)-(N+16,160),15,BF
170 LINE (N,80)-(N+16,160),1,B
180 NEXT N
190 REM Zeichnen der schwarzen
        Klaviertasten
200 FOR N=1 TO 9
210 READ A
220 LINE (A,80)-(A+16,128),1,BF
230 LINE (A,80)-(A+16,128),15,B
240 NEXT N
250 OPEN "grp:"FOR OUTPUT AS #1
260 COLOR 15
270 REM Einzeichnen der Notenerzeuger
        für die weißen Tasten
```

Programm 8   Teil 2
==========

```
280 FOR N=2 TO 14
290 READ A$
300 A=N*16
310 DRAW "bm =a;,168"
320 PRINT #1,A$
330 NEXT N
340 COLOR 1
350 REM Einzeichnen der Notenerzeuger
        für die schwarzen Tasten
360 FOR N=1 TO 9
370 READ A,A$
380 DRAW "bm =a;,64"
390 PRINT #1,A$
400 NEXT N
410 FOR N=1 TO 8
420 REM Einklesen des Sprite-Pfeils
430 READ A$
440 A1$=A1$+CHR$(VAL("&b"+A$))
450 NEXT N
460 SPRITE$(1)=A1$
470 REM Einzeichnen der Notenlinien
        für Einzelton und Akkorde
480 M1=24
490 GOSUB 1230
500 M1=64
510 GOSUB 1230
520 M1=96
530 GOSUB 1230
540 M1=128
550 GOSUB 1230
560 M1=160
570 GOSUB 1230
580 M1=192
590 GOSUB 1230
600 REM Abfrage, ob Taste gedrückt j/n
610 A$=INKEY$
620 IF A$="" GOTO 610
```

Programm 8 Teil 3
==========

```
630 REM weiße Tasten
        Tasatur 'TAB - ü'
640 IF ASC(A$)=9 THEN PUT SPRITE 1,(24,1
35),4,1:A=50:B=4:C=0:N$="O4C":GOSUB 1030
650 IF ASC(A$)=113 THEN PUT SPRITE 1,(40
,135),4,1:A=47:B=0:C=0:N$="O4D":GOSUB 10
30
660 IF ASC(A$)=119 THEN PUT SPRITE 1,(56
,135),4,1:A=44:B=0:C=0:N$="O4E":GOSUB 10
30
670 IF ASC(A$)=101 THEN PUT SPRITE 1,(72
,135),4,1:A=41:B=0:C=0:N$="O4F":GOSUB 10
30
680 IF ASC(A$)=114 THEN PUT SPRITE 1,(88
,135),4,1:A=38:B=0:C=0:N$="O4G":GOSUB 10
30
690 IF ASC(A$)=116 THEN PUT SPRITE 1,(10
4,135),4,1:A=35:B=0:C=0:N$="O4A":GOSUB 1
030
700 IF ASC(A$)=121 THEN PUT SPRITE 1,(12
0,135),4,1:A=32:B=0:C=0:N$="O4B":GOSUB 1
130
710 IF ASC(A$)=117 THEN PUT SPRITE 1,(13
6,135),4,1:A=29:B=0:C=0:N$="O5C":GOSUB 1
130
720 IF ASC(A$)=105 THEN PUT SPRITE 1,(15
2,135),4,1:A=26:B=0:C=0:N$="O5D":GOSUB 1
130
730 IF ASC(A$)=111 THEN PUT SPRITE 1,(16
8,135),4,1:A=23:B=0:C=0:N$="O5E":GOSUB 1
130
740 IF ASC(A$)=112 THEN PUT SPRITE 1,(18
4,135),4,1:A=20:B=0:C=0:N$="O5F":GOSUB 1
130
750 IF ASC(A$)=91 THEN PUT SPRITE 1,(200
,135),4,1:A=17:B=0:C=0:N$="O5G":GOSUB 11
30
```

Programm 8   Teil 4
==========

```
760 IF ASC(A$)=93 THEN PUT SPRITE 1,(216
,135),4,1:A=14:B=4:C=0:N$="O5A":GOSUB 11
30
770 REM schwarze Tasten
        Tastatur '1' - '='
780 IF ASC(A$)=49 THEN PUT SPRITE 1,(32,
100),4,1:A=50:B=4:C=47:N$="O4C#":GOSUB 1
030
790 IF ASC(A$)=50 THEN PUT SPRITE 1,(48,
100),4,1:A=47:B=0:C=44:N$="O4D#":GOSUB 1
030
800 IF ASC(A$)=52 THEN PUT SPRITE 1,(80,
100),4,1:A=41:B=0:C=38:N$="O4F#":GOSUB 1
030
810 IF ASC(A$)=53 THEN PUT SPRITE 1,(96,
100),4,1:A=38:B=0:C=35:N$="O4G#":GOSUB 1
030
820 IF ASC(A$)=54 THEN PUT SPRITE 1,(112
,100),4,1:A=35:B=0:C=32:N$="O4A#":GOSUB
1030
830 IF ASC(A$)=56 THEN PUT SPRITE 1,(144
,100),4,1:A=29:B=0:C=26:N$="O5C#":GOSUB
1130
840 IF ASC(A$)=57 THEN PUT SPRITE 1,(160
,100),4,1:A=26:B=0:C=23:N$="O5D#":GOSUB
1130
850 IF ASC(A$)=45 THEN PUT SPRITE 1,(192
,100),4,1:A=20:B=0:C=17:N$="O5F#":GOSUB
1130
860 IF ASC(A$)=61 THEN PUT SPRITE 1,(208
,100),4,1:A=17:B=0:C=14:N$="O5G#":GOSUB
1130
870 GOTO 610
880 REM DATAs für Bildschirmanzeige
        der Tastatur
890 DATA 32,48,80,96,112,144,160,192,208
900 DATA T,Q,W,E,R,T,Y,U,I,O,P,Ä,Ü
```

Programm 8   Teil 5
==========

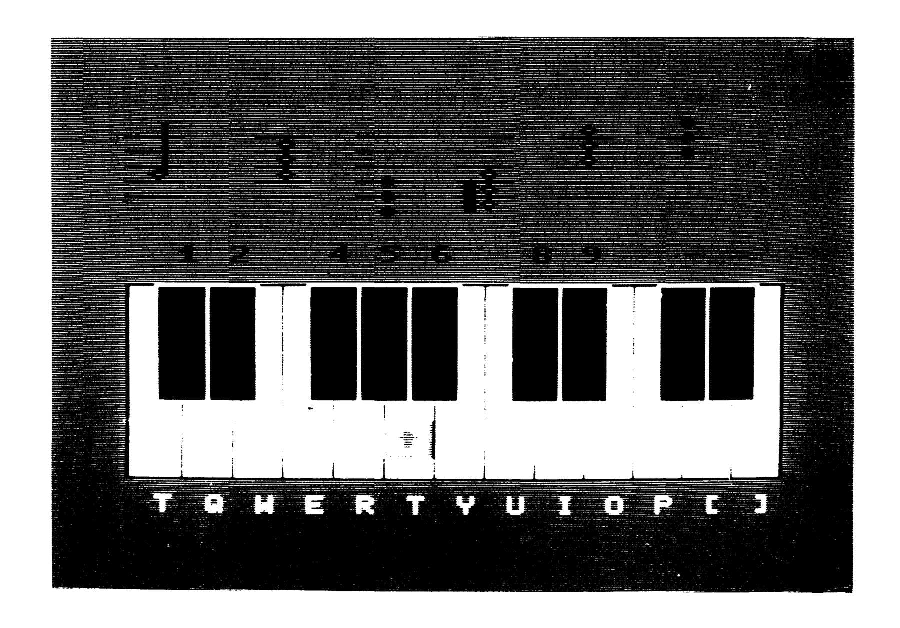
```
910 DATA 40,1,56,2,88,4,104,5,120,6,152,
8,168,9,200,-,216,=
920 REM Spritemuster 8*8 für Pfeil
930 DATA 11111111
940 DATA 10000001
950 DATA 10011001
960 DATA 10111101
970 DATA 10011001
980 DATA 10011001
990 DATA 10000001
1000 DATA 11111111
1010 GOTO 1010
1020 REM Akkordeingabe (AK) j/n; wenn
        nein, dann gilt die Eingabe und
        Anzeige für lediglich eine Note
        mit Notenhals nach oben
1030 IF AK>0 THEN GOTO 1550
1040 LINE(24,10)-(40,54),4,BF
1050 PLAY N$
1060 M1=24
1070 GOSUB 1230
1080 DRAW "bm 35,=a;h2g2f2e2nl=b;u20"
1090 REM schwarze Taste bedeutet: '#'
1100 IF C<>0 THEN DRAW "bm25,=c;":PRINT
#1,"#"
1110 RETURN
1120 REM Akkordeingabe (AK) j/n; wenn
        nein, dann gilt die Eingabe und
        Anzeige für lediglich eine Note
        mit Notenhals nach unten
1130 IF AK>0 THEN GOTO 1550
1140 LINE(24,10)-(40,54),4,BF
1150 PLAY N$
1160 M1=24
1170 GOSUB 1230
1180 DRAW "bm 35,=a;e2f2g2h2nr=b;d20"
1190 REM schwarze Taste bedeutet: '#'
```

Programm 8   Teil 6
==========

```
1200 IF C<>0 THEN DRAW "bm28,=c;":PRINT
#1,"#"
1210 RETURN
1220 REM Einzeichnen der Notenlinien
1230 FOR N=20 TO 44 STEP 6
1240 LINE (M1,N)-(M1+16,N),1
1250 NEXT N
1260 RETURN
1270 REM Unterprogramme für F1 bis F5
        der Akkord wird gespielt
1280 PLAY A$(1),A$(2),A$(3)
1290 RETURN
1300 PLAY B$(1),B$(2),B$(3)
1310 RETURN
1320 PLAY C$(1),C$(2),C$(3)
1330 RETURN
1340 PLAY D$(1),D$(2),D$(3)
1350 RETURN
1360 PLAY E$(1),E$(2),E$(3)
1370 RETURN
1380 REM Einlesen der Akkorde 1 bis 5
        mit vorheriger Löschung des
        letzten Akkords
1390 AK=75:M1=64
1400 GOSUB 1680:GOSUB 1230
1410 RETURN
1420 AK=107:M1=96
1430 GOSUB 1680:GOSUB 1230
1440 RETURN
1450 AK=139:M1=128
1460 GOSUB 1680:GOSUB 1230
1470 RETURN
1480 AK=171:M1=160
1490 GOSUB 1680:GOSUB 1230
1500 RETURN
1510 AK=203:M1=192
1520 GOSUB 1680:GOSUB 1230
```

Programm 8 Teil 7
==========

```
1530 RETURN
1540 REM Die drei Noten für jeden Akkord
         werden zur Anzeige gebracht und
         gespielt
1550 ZZ=ZZ+1
1560 PLAY N$
1570 DRAW "bm=ak;,=a;c1h2g2f2e21=b;"
1580 REM schwarze Taste bedeutet: '#'
1590 IF C<>0 THEN TT=AK-10:DRAW "bm=tt;,
=c;":PRINT #1,"#"
1600 IF AK=75 THEN A$(ZZ)=N$
1610 IF AK=107 THEN B$(ZZ)=N$
1620 IF AK=139 THEN C$(ZZ)=N$
1630 IF AK=171 THEN D$(ZZ)=N$
1640 IF AK=203 THEN E$(ZZ)=N$
1650 IF ZZ=3 THEN ZZ=0:AK=0
1660 RETURN
1670 REM Löschung des alten Akkords
1680 LINE(AK-11,10)-(AK-11+16,54),4,BF
1690 RETURN
```



Kurzbeschreibung zu Programm 9

Ihr MSX-Computer bietet durch den eingebauten Tonchip nicht nur die Möglichkeit, einzelne Töne zu erzeugen, sondern sogar durch drei Kanäle gleichzeitig zu spielen.

Sie haben die Möglichkeit, diese Klänge durch Direktansteuerung der einzelnen Register im Tonchip oder durch den MSX-BASIC-Befehl PLAY zu erzeugen.

Jedoch ist es für Sie manchmal gar nicht so einfach, durch Anwendung des PLAY-Befehls etwas kompliziertere Klänge zu erzeugen, ohne vorher im Handbuch nachgeblättert zu haben, wie die einzelnen Parameter des PLAY-Befehls zu gebrauchen sind.

Programm 9 nimmt Ihnen diese Arbeit ab: Sie können sich durch Eingabe des entsprechenden Buchstaben (z.B. 'o' für Oktave oder 'v' für Lautstärke) die PLAY-Progammierung insofern erleichtern, als daß Sie nun mit der linken und rechten Pfeiltaste den Zahlwert ändern. Hierbei wacht das Programm automatisch darüber, daß keine Falscheingaben entstehen (z.B. Oktave größe als '8').

Wollen Sie den Synthesizer einstellen (Hüllkurvenveränderung mit dem PLAY-Parameter 's'), müssen Sie ihn zuvor durch Drücken der Tasten CTRL-s aktivieren (bzw. später auch wieder durch diese Tastenkombination ausschalten).

Töne erklingen nach Drücken der ENTER-Taste.

Schließlich können Sie Ihre Eingaben nach Wunsch auch so abspeichern, daß daraus DATA-Zeilen zum späteren EinMERGEN in bestehende BASIC-Programme entstehen. Zu der Aktivierung dieser Abspeicherung müssen Sie zu Beginn einen Programmnamen eingeben und jeweils zur Abspeicherung eines Klangs die ESC-Taste drükken. Das Programm und die Datenspeicherung werden abgeschlossen durch die Tastenkombination CTRL-STOP.

Programm 9   Teil 1

```
10 REM Programm 9
20 REM MSX Grafik und Sound
30 REM Copyright 1985 DATA BECKER &
                        Rainer Lüers
40 REM Programm, um mit dem PLAY-Be-
       fehl-Editor nicht nur bequem
       Sounds auszuprobieren, sondern
       diese zudem zum MERGEn in ande-
       ren Programmen später wieder
       verwenden zu können
50 REM Unterbrechungsschutz, damit
       DATAfiles abgeschlossen werden
60 ON STOP GOSUB 1100
70 STOP ON
80 SCREEN 1,1
90 REM Abspeicherung  von
       PLAY-Strings j/n?
100 INPUT "PLAY in DATA (J/ ) ";F$
110 IF F$<>"" THEN INPUT "Name ";F$:TT=9
90:F$="cas:"+F$:OPEN F$ FOR OUTPUT AS #1
120 CLS
130 KEY OFF
140 COLOR 15,4,1
150 REM 8*8-Sprites werden mit dem
        normalen Zeichensatz geladen
160 VDP(6)=VDP(4)
170 REM Anfangswerte für die ver-
        schiedenen PLAY-Parameter
180 O=4
190 L=4
200 R=64
210 V=8
220 S=13
230 M=200
240 REM Hüllkurve 'ein':SS=1
        Hüllkurve 'aus':SS=0
250 SS=0
```

Programm 9   Teil 2
==========

```
260 REM Zeichnen der Notennamen auf
        den Bildschirm
270 PUT SPRITE 0,(46,10),1,99
280 PUT SPRITE 1,(94,10),1,100
290 PUT SPRITE 2,(140,10),1,101
300 PUT SPRITE 3,(186,10),1,102
310 PUT SPRITE 4,(46,60),1,103
320 PUT SPRITE 5,(94,60),1,97
330 PUT SPRITE 6,(140,60),1,98
340 PUT SPRITE 7,(186,60),1,35
350 REM Zeichnen von o(ktave)
                     l(änge)
                     r(est=Pause)
                     v(olume=Lautst.)
360 PUT SPRITE 8,(46,110),15,111
370 PUT SPRITE 9,(94,110),15,108
380 PUT SPRITE 10,(140,110),15,114
390 PUT SPRITE 11,(186,110),15,118
400 REM Zeichnen von s=Hüllkurve
                     m=Hüll.k.feinabst.
410 PUT SPRITE 12,(94,160),10,115
420 PUT SPRITE 13,(140,160),10,109
430 REM Eingabeabfrage
440 A$=INKEY$
450 IF A$="" THEN GOTO 440
460 REM was ist zu tun, wenn eine Note
        angewählt wird?
470 IF A$="c" THEN PUT SPRITE 14,(46,26)
,1,126:N$="c":GOTO 440
480 IF A$="d" THEN PUT SPRITE 14,(94,26)
,1,126:N$="d":GOTO 440
490 IF A$="e" THEN PUT SPRITE 14,(140,26
),1,126:N$="e":GOTO 440
500 IF A$="f" THEN PUT SPRITE 14,(186,26
),1,126:N$="f":GOTO 440
510 IF A$="g" THEN PUT SPRITE 14,(46,76)
,1,126:N$="g":GOTO 440
```

Programm 9   Teil 3

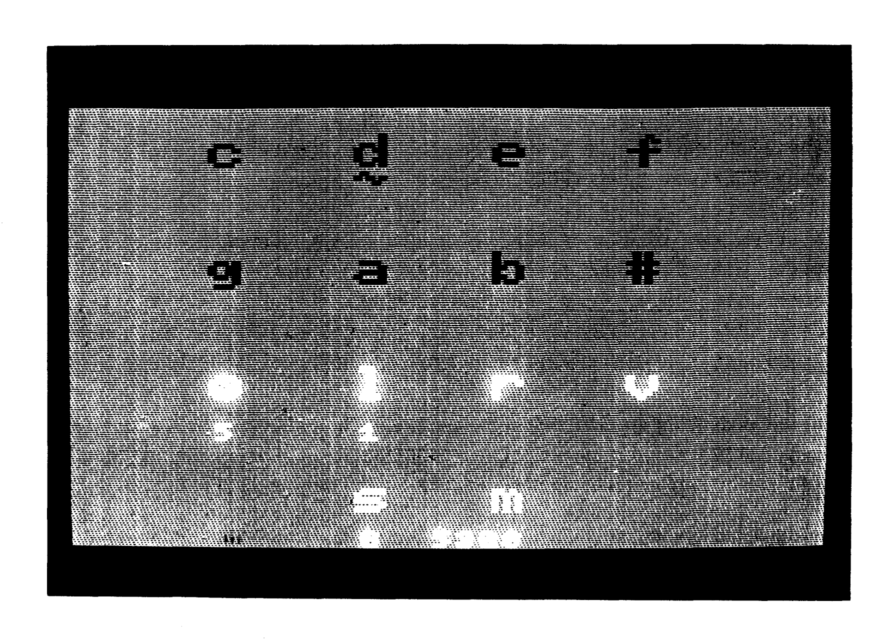
```
520 IF A$="a" THEN PUT SPRITE 14,(94,76)
,1,126:N$="a":GOTO 440
530 IF A$="b" THEN PUT SPRITE 14,(140,76
),1,126:N$="b":GOTO 440
540 IF A$="#" THEN PUT SPRITE 14,(186,76
),1,126:N$=N$+"#":GOTO 440
550 REM was ist zu tun, wenn o=Oktave
        angewählt wird?
560 IF A$="o" THEN PUT SPRITE 14,(46,126
),15,126 ELSE GOTO 630
570 IF O=8 THEN O=0
580 O=O+1
590 LOCATE 4,17
600 PRINT USING "#";O;
610 GOTO 440
620 REM was ist zu tun, wenn l=Länge
        angewählt wird?
630 IF A$="l" THEN PUT SPRITE 14,(94,126
),15,126 ELSE GOTO 700
640 IF L=64 THEN L=0
650 L=L+1
660 LOCATE 9,17
670 PRINT USING "##";L;
680 GOTO 440
690 REM was ist zu tun, wenn r=Pause
        angewählt wird?
700 IF A$="r" THEN PUT SPRITE 14,(140,12
6),15,126 ELSE GOTO 770
710 IF R=64 THEN R=0
720 R=R+1
730 LOCATE 15,17
740 PRINT USING "##";R;
750 GOTO 440
760 REM was ist zu tun, wenn v=Lautst.
        angewählt wird?
770 IF A$="v" THEN PUT SPRITE 14,(186,12
6),15,126 ELSE GOTO 840
```

Programm 9   Teil 4
==========

```
780 IF V=15 THEN V=-1
790 V=V+1
800 LOCATE 21,17
810 PRINT USING "##";V;
820 GOTO 440
830 REM was ist zu tun, wenn s=Hüll-
        kurve angewählt wird?
840 IF A$="s" THEN PUT SPRITE 14,(94,176
),10,126 ELSE GOTO 910
850 IF S=14 THEN S=0
860 S=S+1
870 LOCATE 9,23
880 PRINT USING "##";S;
890 GOTO 440
900 REM was ist zu tun, wenn m=Hüll-
        kurvenabst. angewählt wird?
910 IF A$="m" THEN PUT SPRITE 14,(140,17
6),10,126 ELSE GOTO 960
920 IF M=65500! THEN M=0
930 M=M+100
940 LOCATE 12,23
950 PRINT USING "#####";M;
960 IF A$="M" THEN PUT SPRITE 14,(140,17
6),10,126 ELSE GOTO 1030
970 IF M=100 THEN M=65600!
980 M=M-100
990 LOCATE 12,23
1000 PRINT USING "#####";M;
1010 GOTO 440
1020 REM was ist zu tun, wenn CTRL S=
         An-/Ausschalter Hüllkurve
         angewählt wird?
1030 IF ASC(A$)=19 THEN IF SS=0 THEN SS=
1 ELSE SS=0:GOTO 440
1040 REM was ist zu tun, wenn ENTER=
         Erklingen des Tons
         angewählt wird?
```

Programm 9   Teil 5
==========

```
1Ø5Ø IF ASC(A$)=13 THEN IF SS=1 THEN PLA
Y "o=o;l=l;r=r;v=v;s=s;m=m;xn$;" ELSE PL
AY "o=o;l=l;r=r;v=v;xn$;"
1Ø6Ø REM was ist zu tun, wenn ESC=
         Abspeichern des PLAY-Zustandes
         angewählt wird?
1Ø7Ø IF ASC(A$)=27 AND TT>Ø THEN TT=TT+1
Ø:A1$=STR$(TT)+" DATA "+"o"+STR$(O)+"l"+
STR$(L)+"r"+STR$(R):IF SS=Ø THEN A1$=A1$
+"v"+STR$(V)+N$:PRINT #1,A1$ ELSE A1$=A1
$+"s"+STR$(S)+"m"+STR$(M)+N$:PRINT #1,A1
$
1Ø8Ø GOTO 44Ø
1Ø9Ø REM Unterprogramm für Programm-
         ende mit CTRL-STOP
11ØØ CLOSE
111Ø SCREEN Ø
112Ø END
```



Kurzbeschreibung zu Programm 10

Wer das DATA-BECKER-Buch 'MSX-Programmsammlung' bereits besitzt, dem wird Programm 10 bereits in großen Teilen bekannt sein. Schauen wir uns die Leistungen dieses Programms im einzelnen an:

Ihr MSX-Computer bietet durch den eingebauten Tonchip nicht nur die Möglichkeit, einzelne Töne zu erzeugen, sondern sogar diese Töne durch drei Kanäle gleichzeitig zu spielen.

Sie haben die Möglichkeit, diese Klänge durch Direktansteuerung der einzelnen Register im Tonchip oder durch den MSX-BASIC-Befehl PLAY zu erzeugen.

Jedoch ist es gar nicht so leicht, mit dem SOUND-Befehl Ihrem MSX-Computer Klänge zu entlocken, denn hier reicht es nicht aus, lediglich einen Musikkanal anzusteuern; sie müssen auch das Lautstärke- und das Kombinationsregister mit passenden Zahlwerten belegen, bis schließlich ein Ton erklingt. Außerdem müssen Sie zum Erzeugen eines verzerrten Klangs vorher die Lautstärke zumindest eines Tonkanals auf '16' setzen.

Sicherlich haben Sie bisher schon viel probiert, schließlich aber vielleicht frustriert aufgegeben.

Mit Programm 10 können Sie lediglich durch Gebrauch der Pfeiltasten die Register des Soundchips nicht nur anwählen (Pfeiltasten oben und unten), sondern auch die Werte in den Registern ändern (Pfeiltasten rechts und links). Durch Drücken der Leertaste geben Sie das Kommando zur Tonausgabe.

Schließlich können Sie Ihre Eingaben nach Wunsch auch so abspeichern, daß daraus DATA-Zeilen zum späteren EinMERGEN in bestehende BASIC-Programme entstehen. Zu der Aktivierung dieser Abspeicherung müssen Sie zu Beginn einen Programmnamen eingeben und jeweils zur Abspeicherung eines Klangs die ESC-Taste drükken. Das Programm und die Datenspeicherung werden abgeschlossen durch die Tastenkombination CTRL-STOP.

Programm 10 Teil 1

```
10 REM Programm 10
20 REM MSX Grafik und Sound
30 REM Copyright 1985 DATA BECKER &
                      Rainer Lüers
40 REM Dieser Soundeditor hilft Ihnen
       dabei, die von Ihnen kreierten
       SOUNDs nicht nur leicht einzu-
       geben (bitte Pfeiltasten drük-
       ken), sondern diese auch zu
       hören (Leertaste)
50 REM und, wenn Sie wollen, auch
       für spätere Programme zum
       ReinMERGEn in DATA-Zeilen ab-
       zuspeichern
60 REM Vorbedingungen treffen
70 REM Unterbrechubngsschutz, damit
       DATAfiles abgeschlossen werden
80 SCREEN 0:WIDTH 35
90 CLEAR 2000:DIM A$(21),A(21)
100 ON STOP GOSUB 1280
110 STOP ON
120 KEY OFF
130 COLOR 15,1,1
140 REM Abspeicherung von
        SOUND-Parametern j/n?
150 INPUT "SOUND in DATA (J/ ) ";F$
160 IF F$<>"" THEN INPUT "Name ";F$:TT=9
90:F$="cas:"+F$:OPEN F$ FOR OUTPUT AS #1
170 CLS
180 REM Alle Soundparameter in die
        Ursprungslage versetzen
190 FOR N=0 TO 13
200 SOUND N,0
210 NEXT N
220 SOUND 7,63
230 REM Bildschirmaufbau
240 PRINT TAB(12) "Soundeditor"
```

Programm 10   Teil 2
===========

```
250 PRINT TAB(12) "==========="
260 PRINT
270 PRINT TAB(3);:A$(3)="Tonhoehe fein K
anal 1:":PRINT A$(3)
280 PRINT TAB(3);:A$(4)="Tonhoehe grob K
anal 1:":PRINT A$(4)
290 PRINT TAB(3);:A$(5)="Tonhoehe fein K
anal 2:":PRINT A$(5)
300 PRINT TAB(3);:A$(6)="Tonhoehe grob K
anal 2:":PRINT A$(6)
310 PRINT TAB(3);:A$(7)="Tonhoehe fein K
anal 3:":PRINT A$(7)
320 PRINT TAB(3);:A$(8)="Tonhoehe grob K
anal 3:":PRINT A$(8)
330 PRINT
340 PRINT TAB(6);:A$(10)="Kanal 1: 0/1=T
/2=R:":PRINT A$(10)
350 PRINT TAB(6);:A$(11)="Kanal 2: 0/1=T
/2=R:":PRINT A$(11)
360 PRINT TAB(6);:A$(12)="Kanal 3: 0/1=T
/2=R:":PRINT A$(12)
370 PRINT
380 PRINT TAB(5);:A$(14)="Lautstaerke Ka
nal 1:":PRINT A$(14)
390 PRINT TAB(5);:A$(15)="Lautstaerke Ka
nal 2:":PRINT A$(15)
400 PRINT TAB(5);:A$(16)="Lautstaerke Ka
nal 3:":PRINT A$(16)
410 PRINT
420 PRINT TAB(11);:A$(18)="Rauschperiode
:":PRINT A$(18)
430 PRINT TAB(1);:A$(19)="Huellkurvenper
iode fein:":PRINT A$(19)
440 PRINT TAB(1);:A$(20)="Huellkurvenper
iode grob:":PRINT A$(20)
450 PRINT TAB(14);:A$(21)="Huellkurve:":
PRINT A$(21)
```

Programm 10   Teil 3
===========

```
460 IF TT<>0 THEN PRINT:PRINT "Abspeiche
rung in DATA - Zeilen";
470 REM Wertzuweisung der Sound-
        parameter auf dem Bildschirm
480 FOR N=3 TO 8
490 LOCATE 26,N
500 PRINT A
510 NEXT N
520 FOR N=10 TO 12
530 LOCATE 26,N
540 PRINT A
550 NEXT N
560 FOR N=14 TO 16
570 LOCATE 26,N
580 PRINT A
590 NEXT N
600 FOR N=18 TO 21
610 LOCATE 26,N
620 PRINT A
630 NEXT N
640 REM Einbringen des Zeilencursors
        in die entsprechende Bild-
        schirmzeile in Position 0
650 LOCATE 0,3:PRINT CHR$(200);
660 REM Warten auf Eingabe; die Ein-
        gabemoeglichkeit bezieht sich
        nur auf das Cursorsteuerfeld
        d.h. Pfeil nach oben bzw.
             Pfeil nach unten ->
             Cursorpositionierung
670 REM      Pfeil nach links bzw.
             Pfeil nach rechts ->
             Soundparameter +/-
             Leertaste -> SOUND
             ESC -> Abspeicherung
680 A$=INKEY$:IF A$="" THEN GOTO 680
690 REM was ist zu tun, wenn ESC=
```

Programm 10 Teil 4
===========

Abspeichern der SOUND-Parameter
angewählt wird?

```
700 IF ASC(A$)=27 AND TT>0 THEN TT=TT+10
:A1$=STR$(TT)+" DATA "+STR$(A(3))+","+ST
R$(A(4))+","+STR$(A(5))+","+STR$(A(6))+"
,"+STR$(A(7))+","+STR$(A(8))+","+STR$(A(
18))+","+STR$(A) ELSE GOTO 730
710 A1$=A1$+","+STR$(A(14))+","+STR$(A(1
5))+","+STR$(A(16))+","+STR$(A(19))+","+
STR$(A(20))+","+STR$(A(21))
720 PRINT #1,A1$
730 IF A$=" " THEN SOUND 13,A(21):GOTO 6
80
740 IF ASC(A$)=30 OR ASC(A$)=31 THEN GOT
O 750 ELSE IF ASC(A$)=28 OR ASC(A$)=29 T
HEN GOTO 830 ELSE GOTO 680
750 IF ASC(A$)=30 AND CSRLIN=3 THEN GOTO
 680
760 IF ASC(A$)=31 AND CSRLIN=21 THEN GOT
O 680
770 IF ASC(A$)=31 THEN IF CSRLIN=8 OR CS
RLIN=12 OR CSRLIN=16 THEN A=2 ELSE A=1 E
LSE GOTO 800
780 LOCATE 0,CSRLIN:PRINT " ";:LOCATE 0,
CSRLIN+A:PRINT CHR$(200);:LOCATE 25,CSRL
IN
790 GOTO 680
800 IF ASC(A$)=30 THEN IF CSRLIN=18 OR C
SRLIN=14 OR CSRLIN=10 THEN A=-2 ELSE A=-
1 ELSE GOTO 680
810 LOCATE 0,CSRLIN:PRINT " ";:LOCATE 0,
CSRLIN+A:PRINT CHR$(200);:LOCATE 25,CSRL
IN
820 GOTO 680
830 IF ASC(A$)=28 THEN Z=1 ELSE Z=-1
840 LOCATE 26,CSRLIN:PRINT "   ";:LOCATE
 26,CSRLIN
```

Programm 1Ø   Teil 5
===========

```
85Ø REM Eingeben der Soundparameter
        + oder - in den 'SOUND'-Be-
        fehl; ausserdem Ueberpruefung
        auf Erreichen der Grenzwerte
        sowie Kontrollieren der
        Bildschirmanzeige
86Ø REM Tonhoehe Kanal 1 fein
        =====================
87Ø IF CSRLIN=3 THEN IF (A(3)+Z<Ø OR A(3
)+Z>255) THEN PRINT A(3);:GOTO 68Ø ELSE
A(3)=A(3)+Z:PRINT A(3);:SOUND Ø,A(3):GOT
O 68Ø
88Ø REM Tonhoehe Kanal 1 grob
        =====================
89Ø IF CSRLIN=4 THEN IF (A(4)+Z<Ø OR A(4
)+Z>15) THEN PRINT A(4);:GOTO 68Ø ELSE A
(4)=A(4)+Z:PRINT A(4);:SOUND 1,A(4):GOTO
 68Ø
9ØØ REM Tonhoehe Kanal 2 fein
        =====================
91Ø IF CSRLIN=5 THEN IF (A(5)+Z<Ø OR A(5
)+Z>255) THEN PRINT A(5);:GOTO 68Ø ELSE
A(5)=A(5)+Z:PRINT A(5);:SOUND 2,A(5):GOT
O 68Ø
92Ø REM Tonhoehe Kanal 2 grob
        =====================
93Ø IF CSRLIN=6 THEN IF (A(6)+Z<Ø OR A(6
)+Z>15) THEN PRINT A(6);:GOTO 68Ø ELSE A
(6)=A(6)+Z:PRINT A(6);:SOUND 3,A(6):GOTO
 68Ø
94Ø REM Tonhoehe Kanal 3 fein
        =====================
95Ø IF CSRLIN=7 THEN IF (A(7)+Z<Ø OR A(7
)+Z>255) THEN PRINT A(7);:GOTO 68Ø ELSE
A(7)=A(7)+Z:PRINT A(7);:SOUND 4,A(7):GOT
O 68Ø
96Ø REM Tonhoehe Kanal 3 grob
```

Programm 1Ø   Teil 6
===========

```
          ====================
97Ø IF CSRLIN=8 THEN IF (A(8)+Z<Ø OR A(8
)+Z>15) THEN PRINT A(8);:GOTO 68Ø ELSE A
(8)=A(8)+Z:PRINT A(8);:SOUND 5,A(8):GOTO
 68Ø
98Ø REM Lautstaerke Kanal 1
        ====================
99Ø IF CSRLIN=14 THEN IF (A(14)+Z<Ø OR A
(14)+Z>16) THEN PRINT A(14);:GOTO 68Ø EL
SE A(14)=A(14)+Z:PRINT A(14);:SOUND 8,A(
14):GOTO 68Ø
1ØØØ REM Lautstaerke Kanal 2
         ====================
1Ø1Ø IF CSRLIN=15 THEN IF (A(15)+Z<Ø OR
A(15)+Z>16) THEN PRINT A(15);:GOTO 68Ø E
LSE A(15)=A(15)+Z:PRINT A(15);:SOUND 9,A
(15):GOTO 68Ø
1Ø2Ø REM Lautstaerke Kanal 3
         ====================
1Ø3Ø IF CSRLIN=16 THEN IF (A(16)+Z<Ø OR
A(16)+Z>16) THEN PRINT A(16);:GOTO 68Ø E
LSE A(16)=A(16)+Z:PRINT A(16);:SOUND 1Ø,
A(16):GOTO 68Ø
1Ø4Ø REM Rauschperiode
         =============
1Ø5Ø IF CSRLIN=18 THEN IF (A(18)+Z<Ø OR
A(18)+Z>31) THEN PRINT A(18);:GOTO 68Ø E
LSE A(18)=A(18)+Z:PRINT A(18);:SOUND 6,A
(18):GOTO 68Ø
1Ø6Ø REM Huellkurvenperiode fein
         ========================
1Ø7Ø IF CSRLIN=19 THEN IF (A(19)+Z<Ø OR
A(19)+Z>255) THEN PRINT A(19);:GOTO 68Ø
ELSE A(19)=A(19)+Z:PRINT A(19);:SOUND 11
,A(19):GOTO 68Ø
1Ø8Ø REM Huellkurvenperiode grob
         ========================
```

Programm 10   Teil 7

```
1090 IF CSRLIN=20 THEN IF (A(20)+Z<0 OR
A(20)+Z>255) THEN PRINT A(20);:GOTO 680
ELSE A(20)=A(20)+Z:PRINT A(20);:SOUND 12
,A(20):GOTO 680
1100 REM Huellkurve
         ==========
1110 IF CSRLIN=21 THEN IF (A(21)+Z<0 OR
A(21)+Z>15) THEN PRINT A(21);:GOTO 680 E
LSE A(21)=A(21)+Z:PRINT A(21);:SOUND 13,
A(21):GOTO 680
1120 REM Kanal 1: aus/Ton/Rau/beides
         ===========================
1130 IF CSRLIN=10 THEN IF (A(10)+Z<0 OR
A(10)+Z>3) THEN PRINT A(10);:GOTO 680 EL
SE A(10)=A(10)+Z:PRINT A(10);
1140 REM Kanal 2: aus/Ton/Rau/beides
         ===========================
1150 IF CSRLIN=11 THEN IF (A(11)+Z<0 OR
A(11)+Z>3) THEN PRINT A(11);:GOTO 680 EL
SE A(11)=A(11)+Z:PRINT A(11);
1160 REM Kanal 3: aus/Ton/Rau/beides
         ===========================
1170 IF CSRLIN=12 THEN IF (A(12)+Z<0 OR
A(12)+Z>3) THEN PRINT A(12);:GOTO 680 EL
SE A(12)=A(12)+Z:PRINT A(12);
1180 IF CSRLIN<10 OR CSRLIN>12 THEN GOTO
 1260
1190 REM genaue Berechnung des 'SOUND'-
         Parameters 7
1200 A=0
1210 IF A(10)=1 THEN A=A+8 ELSE IF A(10)
=2 THEN A=A+1 ELSE IF A(10)=0 THEN A=A+1
+8
1220 IF A(11)=1 THEN A=A+16 ELSE IF A(11
)=2 THEN A=A+2 ELSE  IF A(11)=0 THEN A=A
+16+2
1230 IF A(12)=1 THEN A=A+32 ELSE IF A(12
```

Programm 10   Teil 8
===========

```
)=2 THEN A=A+4 ELSE IF A(12)=0 THEN A=A+
32+4
1240 A=A+128
1250 SOUND 7,A
1260 GOTO 680
1270 WIDTH 40
1280 REM Unterprogramm für Programmende
        mit CTRL STOP
1290 CLOSE
1300 SCREEN 0
1310 END
```

```
                 Soundeditor
                 ===========

    Tonhoehe fein Kanal 1:   107
    Tonhoehe grob Kanal 1:   13
    Tonhoehe fein Kanal 2:   2
    Tonhoehe grob Kanal 2:   9
    Tonhoehe fein Kanal 3:   150
    Tonhoehe grob Kanal 3:   6

       Kanal 1: 0/1=T/2=R:   1
       Kanal 2: 0/1=T/2=R:   2
       Kanal 3: 0/1=T/2=R:   1

      Lautstaerke Kanal 1:   16
      Lautstaerke Kanal 2:   7
      Lautstaerke Kanal 3:   12

            Rauschperiode:   2
 Huellkurvenperiode fein:   19
 Huellkurvenperiode grob:   1
               Huellkurve:   8

 Abspeicherung in DATA - Zeilen
```

Kurzbeschreibung zu Programm 11

Wie Sie in diesem Buch nachlesen können, bietet Ihr MSX-Computer eine sehr große Anzahl von Grafikbefehlen.

Wollen Sie diese Grafikbefehle zur Erstellung eines Bildes gebrauchen, müssen Sie sich voher auf einem Millimeterpapier die einzelnen Positionen zur Grafikeingabe ausrechnen, was leider unvermeidlich ist, zur Erstellung eines Titelbildes jedoch mit Programm 11 keine große Schwierigkeit mehr bedeuten sollte.

Sie bewegen sich mit einem Fadenkrez auf dem Bildschirm und steuern dadurch, ohne sich um Koordinaten kümmern zu müssen, den entsprechenden Zielpunkt an. Anschließend haben Sie in diesem Programm die Möglichkleit durch Drücken der Funktionstasten MSX-BASIC-Befehle direkt auszuführen:

F1: Kreis zeichen	F2: Linie zeichnen
F3: Zeichenfarbe ändern	F4: Text auf Grafik
F5: Zeichnen ja/nein	F6: Abspeichern des Bildes
F7: Laden eines Bildes	F8: Umrandungsfarbe ändern
F9: Ausmalen von Flächen	F10: Kopieren von Teilen

Sie müssen sich allerdings damit abfinden, daß die Abspeicherung und Ladung von Bildschirmen nicht in Sekundenbruchteilen sondern in Minutenzeiträumen abläuft.

Programm 11   Teil 1

```
10 REM Programm 11
20 REM MSX Grafik und Sound
30 REM Copyright 1985 DATA BECKER &
                      Rainer Lüers
40 REM Editor, um mit denn zur Verfü-
       gung stehenden Befehlen
       Grafiken auf dem Bildschirm er-
       stellen und abspeichern zu
       können. Bequeme Funktionsta-
       stensteuerung.
50 REM Kassettenabspeicherung der Bild-
       schirme ist ebenso möglich, wie
       das Herauskopieren von Bild-
       schirmteilen. übersicht
       der Befehle siehe Zeile 70 ff.
60 REM Damit man nicht mit einer un-
       vorhersehbaren Farbkombination
       auf den Textbildschirm
       zurueckkehrt Abbruchsicherungs-
       routine mit ON STOP GOSUB
70 ON STOP GOSUB 1990
80 STOP ON
90 REM F1 Kreis zeichnen, wenn der
          zweite Cursor gesetzt ist
       F2 Linie zeichnen, wenn der
          zweite Cursor gesetzt ist
       F3 Zeichenfarbe aendern
          Zweistellige Zahl eingeben
100 REM F4 Text an der Cursorposition
           schreiben; <ENTER> heisst
           Ende
110 REM F5 Zeichnen oder nicht zeichnen
           nicht zeichnen=Cursorbewegung
        F6 Abspeichern mit Name
           und Ueberschrift
           (dauert ca. 10 Minuten)
        F7 Einladen von Bildschirmen
```

Programm 11   Teil 2
===========

```
120 REM    Namenseingabe
        F8 Umrandungsfarbe aendern
           immer mit zwei Ziffern
           eingeben (z.B. '08' aber
           ohne <ENTER>
130 REM F9 Ausmalen mit der
           aktuellen Farbe
           Vorsicht! Es wird bis zur
           gleichen Randfarbe aus-
           gemalt
140 REM F10 Kopieren von Bildschirm-
            teilen. Aktueller Cursor
            ist die Ecke unten links
            2.Cursor=oben rechts
            3.Cursor=neues Bild
              Ecke unten links
150 ON KEY GOSUB 550,680,810,850,930,960
,1190,1410,1440,1470
160 REM Vorbedingungen treffen
170 COLOR 15,4,5
180 SCREEN 2,2
190 FOR N=1 TO 10
200 KEY(N) ON
210 NEXT N
220 REM Einlesen des Fadenkreuzes,
        das als Grafikcursor=Sprite
        dient
230 FOR N=1 TO 8
240 READ A$
250 A$="&b"+A$
260 B$=B$+CHR$(VAL(A$))
270 NEXT  N
280 SPRITE$(1)=B$
290 REM Einlesen des Fadenkreuz-
        mittelpunkts, damit man auf
        jedwedem Untergrund den
        Cursor auch sieht
```

Programm 11   Teil 3
===========

```
300 FOR N=1 TO 8
310 READ A$
320 A$="&b"+A$
330 C$=C$+CHR$(VAL(A$))
340 NEXT  N
350 SPRITE$(2)=C$
360 REM Setzen des Grafikcursors
        in den Bildschirmmittelpunkt
        Farbe schwarz
370 A=128
380 B=96
390 F=1
400 COLOR F
410 PUT SPRITE 1,(A,B),1
420 PUT SPRITE 2,(A,B),15
430 REM Soll waehrend der Cursorbe-
list .-
440 IF ME=1 THEN PSET(A+4,B+4),F
450 REM Abfrage der Richtung ueber
        die Betaetigung der Pfeil-
        tasten
460 A$=INKEY$
470 IF A$="" THEN GOTO 460
480 IF ASC(A$)=28 THEN IF A<>250 THEN A=
A+1
490 IF ASC(A$)=29 THEN IF A<>-4 THEN A=A
-1
500 IF ASC(A$)=30 THEN IF B<>-4 THEN B=B
-1
510 IF ASC(A$)=31 THEN IF B<>186 THEN B=
B+1
520 PUT SPRITE 1,(A,B),1
530 PUT SPRITE 2,(A,B),15
540 GOTO 410
550 REM Kreis zeichnen
        ==============
560 C=A
```

Programm 11   Teil 4
===========

```
570 D=B
580 REM Abfrage der Richtung ueber
        die Betaetigung der Pfeil-
        tasten
590 A$=INKEY$
600 IF A$="" THEN GOTO 590
610 IF ASC(A$)=28 THEN IF C<>250 THEN C=
C+1
620 IF ASC(A$)=29 THEN IF C<>-4 THEN C=C
-1
630 IF ASC(A$)=30 THEN IF D<>-4 THEN D=D
-1
640 IF ASC(A$)=31 THEN IF D<>186 THEN D=
D+1
650 IF (A$)=CHR$(13) THEN A1=ABS(A-C):A2
=ABS(B-D):IF A1>=A2 THEN CIRCLE (A+4,B+4
),A1+13,F,,,4/3:RETURN ELSE CIRCLE (A+4,
B+4),A2,F,,,4/3:RETURN
660 PUT SPRITE 1,(C,D),1
670 GOTO 590
680 REM Linie zeichnen
        ==============
690 C=A
700 D=B
710 REM Abfrage der Richtung ueber
        die Betaetigung der Pfeil-
        tasten
720 A$=INKEY$
730 IF A$="" THEN GOTO 720
740 IF ASC(A$)=28 THEN IF C<>250 THEN C=
C+1
750 IF ASC(A$)=29 THEN IF C<>-4 THEN C=C
-1
760 IF ASC(A$)=30 THEN IF D<>-4 THEN D=D
-1
770 IF ASC(A$)=31 THEN IF D<>186 THEN D=
D+1
```

Programm 11   Teil 5
===========

```
780 IF (A$)=CHR$(13) THEN LINE (A+4,B+4)
-(C+4,D+4),F:RETURN
790 PUT SPRITE 1,(C,D),1
800 GOTO 720
810 REM Farbe aendern
        =============
820 F$=INPUT$(2)
830 F=VAL(F$)
840 IF F<16 THEN COLOR F:RETURN ELSE RET
URN
850 REM Text eingeben
        =============
860 OPEN "grp:"FOR OUTPUT AS #1
870 DRAW "bm =a;,=b;"
880 A$=INKEY$
890 IF A$="" THEN GOTO 880
900 IF A$=CHR$(13) THEN CLOSE:RETURN
910 PRINT #1,A$;
920 GOTO 880
930 REM Zeichnen - nicht zeichnen
        =========================
940 IF ME=1 THEN ME=0 ELSE ME=1
950 RETURN
960 REM Abspeichern
        ===========
970 REM Damit es nicht zu unvorher-
        gesehenen Farbvermischungen
        kommen kann
980 LINE (100,15)-(160,0),1,BF
990 OPEN "grp:" FOR OUTPUT AS #1
1000 NN$=""
1010 COLOR 15
1020 DRAW "bm 108,4"
1030 REM Eingabe des Programmnamens=
         der Bildueberschrift mit bis
         zu sechs Buchstaben/Zeichen
1040 FOR N=1 TO 6
```

Programm 11   Teil 6
===========

```
1Ø5Ø NN$=NN$+N$
1Ø6Ø N$=INPUT$(1)
1Ø7Ø IF N$=CHR$(13) THEN GOTO 1Ø8Ø ELSE
PRINT #1,N$;:NEXT N
1Ø8Ø A$=INKEY$
1Ø9Ø REM Sicherheitsfunktion
11ØØ IF A$="" THEN GOTO 1Ø8Ø ELSE IF A$<
>CHR$(13) THEN RETURN
111Ø CLOSE #1
112Ø OPEN NN$ FOR OUTPUT AS #1
113Ø REM Abspeicherung der 16K
         Video-RAM
114Ø FOR N=Ø TO 16383
115Ø PRINT #1,VPEEK(N)
116Ø NEXT N
117Ø CLOSE
118Ø RETURN
119Ø REM Laden
         =====
12ØØ REM Damit es nicht zu unvorher-
         gesehenen Farbvermischungen
         kommen kann
121Ø LINE (1ØØ,15)-(16Ø,Ø),1,BF
122Ø OPEN "grp:" FOR OUTPUT AS #1
123Ø NN$=""
124Ø COLOR 15
125Ø DRAW "bm 1Ø8,4"
126Ø REM Eingabe des Programmnamens=
         der Bildueberschrift mit bis
         zu sechs Buchstaben/Zeichen
127Ø FOR N=1 TO 6
128Ø NN$=NN$+N$
129Ø N$=INPUT$(1)
13ØØ IF N$=CHR$(13) THEN GOTO 1Ø8Ø ELSE
PRINT #1,N$;:NEXT N
131Ø REM Sicherheitsfunktion
132Ø A$=INKEY$
```

Programm 11   Teil 7
===========

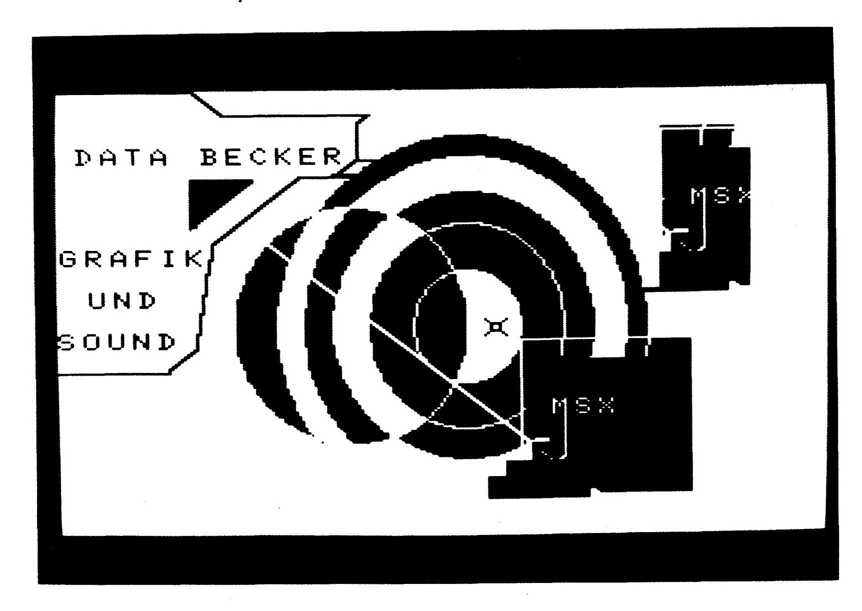
```
1330 IF A$="" THEN GOTO 1320 ELSE IF A$<
>CHR$(13) THEN RETURN
1340 CLOSE #1
1350 REM Einladen der 16K
        Video-RAM
1360 OPEN NN$ FOR INPUT AS #1
1370 FOR N=0 TO 16383
1380 INPUT #1,D
1390 VPOKE N,D
1400 NEXT N
1410 REM Border aendern
        ==============
1420 A$=INPUT$(2)
1430 IF VAL(A$)<16 THEN COLOR ,,VAL(A$):
RETURN ELSE RETURN
1440 REM Ausmalen
        ========
1450 PAINT (A,B),F
1460 RETURN
1470 REM Verdoppeln
        ==========
1480 C1=A
1490 D1=B
1500 REM Festlegen des Ursprung-
        bildschirmausschnitts
        Ecke rechts oben
1510 REM Abfrage der Richtung ueber
        die Betaetigung der Pfeil-
        tasten
1520 A$=INKEY$
1530 IF A$="" THEN GOTO 1520
1540 IF ASC(A$)=28 THEN IF C1<>250 THEN
C1=C1+1
1550 IF ASC(A$)=29 THEN IF C1<>-4 THEN C
1=C1-1
1560 IF ASC(A$)=30 THEN IF D1<>-4 THEN D
1=D1-1
```

Programm 11   Teil 8

```
1570 IF ASC(A$)=31 THEN IF D1<>186 THEN
D1=D1+1
1580 IF (A$)=CHR$(13) THEN GOTO 1620
1590 PUT SPRITE 1,(C1,D1),1
1600 GOTO 1520
1610 REM Der Cursor muss im Verhaelt-
         nis zum Ursprung rechts oben
         postiert werden!
1620 IF A>C1 OR B<D1 THEN BEEP:RETURN EL
SE C2=A:D2=B
1630 REM Festlegen der Kopie
         Bildschirmausschnitt
         Ecke links unten
1640 REM Abfrage der Richtung ueber
         die Betaetigung der Pfeil-
         tasten
1650 A$=INKEY$
1660 IF A$="" THEN GOTO 1650
1670 IF ASC(A$)=28 THEN IF C2<>250 THEN
C2=C2+1
1680 IF ASC(A$)=29 THEN IF C2<>-4 THEN C
2=C2-1
1690 IF ASC(A$)=30 THEN IF D2<>-4 THEN D
2=D2-1
1700 IF ASC(A$)=31 THEN IF D2<>186 THEN
D2=D2+1
1710 IF (A$)=CHR$(13) THEN GOTO 1750
1720 PUT SPRITE 1,(C2,D2),1
1730 GOTO 1650
1740 REM Eigentlicher Kopiervorgang
         vom Ursprungsbild ausgehend
1750 FOR N=A TO C1
1760 FOR M=B TO D1 STEP-1
1770 PSET(C2+(N-A),D2+(M-B)),POINT(N,M)
1780 NEXT M,N
1790 RETURN
1800 REM Sprite 1
```

Programm 11   Teil 9
===========

```
         Schwarzes Fadenkreuz
1810 DATA 10000001
1820 DATA 01000010
1830 DATA 00111100
1840 DATA 00100100
1850 DATA 00100100
1860 DATA 00111100
1870 DATA 01000010
1880 DATA 10000001
1890 REM Sprite 2
         Weisser Fadenkreuzmittel-
         punkt
1900 DATA 00000000
1910 DATA 00000000
1920 DATA 00000000
1930 DATA 00011000
1940 DATA 00011000
1950 DATA 00000000
1960 DATA 00000000
1970 DATA 00000000
1980 REM Bei Programmunterbrechung
         wird wieder ein lesbares Farb-
         verhaeltnis erzeugt
1990 COLOR 15,1
```



Kurzbeschreibung zu Programm 12

Weiter hinten im diesem Buch finden Sie den ausgedruckten Zeichensatz Ihres MSX-Computers sowie einen zweiten Zeichensatz, einen sogenannten Computerschrift-Zeichensatz. Letztgenannter Zeichensatz ist zur Programmeingabe in meinem DATA BECKER-Buch 'MSX-Programmsammlung' zu finden.

Da Sie im vorliegenden Buch nicht nur sehr viel über die Abspeicherung und Änderung des Zeichensatzes nachlesen können, sondern Ihnen auch mit Programm 14 ein äußerst luxuriöser Zeichensatzgenerator vorliegt, hielt ich es für sinnvoll, das Ausdruckprogramm ein klein wenig zu erweitern und als Listing in diesem Buch mit anzuführen. So können Sie selbst z.B. den Zeichensatz ändern (mit Programm 14) und anschließend Programm 12 in Ihren MSX-Computer einladen (da Video-RAM und Programmspeicher getrennt sind, wird Ihr Zeichensatz durch das Einladen eines neuen Programms selbstverständlich nicht gelöscht!).

Jedoch achten Sie darauf, daß dieses Programm eigentlich für einen Nicht-MSX-Drucker und zwar den EPSON MX-80 und nicht für einen MSX-Drucker entwickelt wurde. Die einzige Programmänderung, die Sie somit vornehmen müssen (falls Sie Besitzer eines speziellen MSX-Druckers sind: Ändern Sie das EPSON-Blockzeichen CHR$(223) in das MSX-Blockzeichen CHR$(219) ab.

Die Papierlänge ist auf Standard=72 Zeilen pro Seite eingestellt. Sie dürfen zudem bei der Eingabe der auszudruckenden Zeichennummern nur eine Anzahl eingeben, die durch 3 teilbar ist (z.B. Zeichen 129 bis 218 = 90 Zeichen = 30 Zeilen zu je 3 Zeichen = 7.5 Seiten mit je 12 Zeichen).

Übrigens: Sie müssen sich zur Anwendung dieses Programms samt geändertem Zeichensatz bereits auf SCREEN 1 befinden. Bei Anwendung eines SCREEN-Befehls wird sonst der Video-RAM-Speicher und damit auch Ihr Zeichensatz gelöscht.

Programm 12 Teil 1

```
10 REM Programm 12
20 REM MSX Grafik und Sound
30 REM Copyright 1985 DATA BECKER &
                         Rainer Lüers
40 REM Programm, um den Zeichensatz ver-
       größert auszudrucken. Das Pro-
       gramm wurde für einen EPSON-Druk-
       ker (Nicht-MSX) entwickelt und
       muß von Ihnen gegebenenfalls an-
       gepaßt werden (Block bei EPSON:
50 REM CHR$(223), Block bei MSX:
       CHR$(219)). Nach jeweils 12 aus-
       gedruckten Zeichen findet ein
       Seitenvorschub statt. Auch hier
       müssen Sie anpassen, wenn Sie mit
       Nicht-DIN A4-Papier=72 Zeilen/
60 REM Seite arbeiten.
70 CLS
80 PRINT "Sie muessen sich auf"
90 PRINT "SCREEN(1) befinden!"
100 PRINT
110 PRINT "Welches Zeichen soll als"
120 PRINT "erstes ausgedruckt"
130 INPUT "werden (1 bis 255) ";A
140 PRINT
150 PRINT "Welches Zeichen soll als"
160 PRINT "letztes ausgedruckt"
170 PRINT "werden (";RIGHT$(STR$(A),(LEN
(STR$(A))-1));
180 INPUT " bis 255) ";B
190 REM Kontrolle u.a., ob die ausge-
        wählten Zeichen in den Dreier-
        ausdruck hineinpassen
200 IF A>B OR B>255 OR A<0 OR (B+1-A)/3<
>INT((B+1-A)/3) THEN RUN
210 PRINT
220 PRINT "Das sind die Zeichen,"
```

Programm 12   Teil 2
===========

```
230 PRINT "die ausgedruckt"
240 PRINT "werden sollen:"
250 PRINT
260 FOR N=A TO B
270 PRINT CHR$(N);
280 NEXT N
290 PRINT
300 INPUT "Richtig -> <ENTER> ";A$
310 IF A$<>"" THEN RUN
320 REM Speicherplätze bei SCREEN 1, wo
        das Aussehen der CHR$-Zeichen
        abgespeichert ist
330 FOR N=A*8 TO B*8 STEP24
340 LPRINT
350 REM Ausdruck des linken Zeichens
360 LPRINT "Characterzeichen";INT(N/8);
370 REM Ausdruck des mittleren Zeichens
380 LPRINT TAB(29)"Characterzeichen";INT
((N+8)/8);
390 REM Ausdruck des rechten Zeichens
400 LPRINT TAB(58)"Characterzeichen";INT
((N+16)/8)
410 LPRINT
420 LPRINT TAB(8)"76543210";
430 LPRINT TAB(37);"76543210";
440 LPRINT TAB(66);"76543210"
450 LPRINT
460 FOR M=0 TO 7
470 REM Umrechnung von Dezimal- in volle
        8 Bit-Binärdarstellung
480 A$=BIN$(VPEEK(N+M))
490 B$=BIN$(VPEEK(N+M+8))
500 C$=BIN$(VPEEK(N+M+16))
510 A$=STRING$(8-LEN(A$),"0")+A$
520 B$=STRING$(8-LEN(B$),"0")+B$
530 C$=STRING$(8-LEN(C$),"0")+C$
540 LPRINT TAB(5)M;TAB(7);
```

Programm 12   Teil 3
===========

```
550 FOR L=1TO8
560 REM Umrechnung von Binär- in Block-
        Leerzeichen-Darstellung auf dem
        Drucker
570 IF MID$(A$,L,1)="1" THEN LPRINT CHR$
(223);ELSE LPRINT" ";
580 NEXT L
590 LPRINT TAB(34)M;TAB(36);
600 FOR L=1TO8
610 IF MID$(B$,L,1)="1" THEN LPRINT CHR$
(223);ELSE LPRINT" ";
620 NEXT L
630 LPRINT TAB(63)M;TAB(65);
640 FOR L=1TO8
650 IF MID$(C$,L,1)="1" THEN LPRINT CHR$
(223);ELSE LPRINT" ";
660 NEXT L
670 LPRINT
680 NEXT M
690 LPRINT
700 REM Kontrolle, ob ein Seitenvorschub
        bereits erforderlich ist
710 Z=Z+1
720 IF Z=4 THEN Z=0:FOR U=1 TO 16:LPRINT
:NEXT
730 NEXT N
740 RUN
```

```
Sie muessen sich auf
SCREEN(1) befinden!

Welches Zeichen soll als
erstes ausgedruckt
werden (1 bis 255) ? 33

Welches Zeichen soll als
letztes ausgedruckt
werden (33 bis 255) ? 128

Das sind die Zeichen,
die ausgedruckt
werden sollen:

!"#$%&'()*+,-./0123456789:;<=
>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ
[\]^_`abcdefghijklmnopqrstuvw
xyz{|}Ç
Richtig -> <ENTER> ?
```

Kurzbeschreibung zu Programm 13

Ihr MSX-Computer bietet Ihnen 2 Textbildschirme (SCREEN 0 mit 40 Zeichen pro Zeile und SCREEN 1 mit 32 Zeichen pro Zeile) sowie zwei Grafikbildschirme (SCREEN 2 mit 256*192 Punkten und SCREEN 3 mit 64*48 Punkten).

Zwar ist es möglich, auf den Grafikbildschirmen auch Text auszugeben, aber auf einem Textbildschirm Grafik darzustellen, das geht nur so lange, wie es die Spritesteuerung auf SCREEN 1 anbetrifft.

Mit dem vorliegenden Programm wird dieser Trennung Text-Grafik nun endlich ein Ende bereitet - nein, so einfach geht das auch nicht!

Wir haben in Programm 13 ein bisschen getrickst und zwar folgendermaßen: Sie können auf SCREEN 0 bzw. SCREEN 1 (die Textbildschirme) gleichzeitig bis zu 256 verschiedene Zeichen darstellen. Wird ein Zeichen in seinem Aussehen von Ihnen geändert, ändern sich gleichzeitig sämtliche Zeichen mit derselben Zeichencodenummer auf dem Bildschirm ebenfalls. Sie können jedes dieser 256 Zeichen nach eigenem Belieben in seinem Aussehen verändern. Soviel zur Vorbedingung (nachzulesen in den Kapiteln über CHR$ und ASC in diesem Buch).

Mit Programm 13 wird nun eine ständige Änderung einer Anzahl von bis zu 4 Zeichen durchgeführt, so daß Sie die Vorstellung haben, als wenn eine kleine Grafik vor Ihren Augen auf dem Textbildschirm ablaufen würde. Sogar haben wir es schließlich so weit getrieben, daß in 4 Zeichen die Anfangsbuchstaben von DATA BECKER auf den Bildschirm geschrieben werden ...

Programm 13   Teil 1

```
10 REM Programm 13
20 REM MSX Grafik und Sound
30 REM Copyright 1985 DATA BECKER &
                      Rainer Lüers
40 REM Dieses Programm zeigt Ihnen, wie
       Sie auf SCREEN 1 (unter Berück-
       sichtigung der anderen Character-
       zeichen-Startadresse im VRAM auch
       auf SCREEN Ø machbar), der eigent-
50 REM lich bis auf vor- bzw. selbstde-
       finierte Grafikzeichen nicht gra-
       fikfähig ist, doch Grafiken er-
       stellen können. Nur näheren Er-
       klärung dieses Phänomens siehe
60 REM den erläuternden Text zum Pro-
       gramm in diesem Buch
70 REM Vorbereitungen treffen
80 SCREEN 1
90 KEY OFF
100 REM Characterzeichen 250 löschen
110 REM 1) ein Punkt verändert seine
           Position fließend
120 GOSUB 980
130 REM links oben wird CHR$(250) gesetz
t
140 LOCATE Ø,Ø
150 PRINT CHR$(250)
160 LOCATE 6,10
170 PRINT "SCREEN Ø mit HGR 1"
180 FOR L=1 TO 10
190 REM Punktbewegung
200 GOSUB 870
210 REM CHR$-Zeichen 250 löschen
220 GOSUB 980
230 NEXT L
240 REM 2) viele Punkte verändern ihre
           Position gleichmäßig flies-
```

Programm 13   Teil 2
===========

```
              send
250 GOSUB 1300
260 REM auf den gesamten Bildschirm wird
        CHR$-Zeichen 250 gesetzt
270 LOCATE 6,10
280 PRINT "SCREEN 0 mit HGR 1a"
290 FOR L=1 TO 10
300 REM Punktbewegung
310 GOSUB 870
320 REM CHR$-Zeichen 250 löschen
330 GOSUB 980
340 NEXT L
350 REM 3) Zeichnung in einem CHR$-Zei-
           chen gleichmäßig fließend
360 CLS
370 LOCATE 0,0
380 PRINT CHR$(250)
390 LOCATE 6,10
400 PRINT "SCREEN 1 mit HGR 2"
410 FOR L=1 TO 10
420 REM Zeichnungsentwicklung
430 GOSUB 1030
440 REM Zeichen löschen
450 GOSUB 980
460 NEXT L
470 REM 4) Zeichnung in vielen Positio-
           nen gleichmäßig fließend
480 GOSUB 1300
490 REM auf den gesamten Bildschirm
        wird CHR$-Zeichen 250 gesetzt
500 LOCATE 6,10
510 PRINT "SCREEN 0 mit HGR 2a"
520 FOR L=1 TO 10
530 REM Zeichenentwicklung
540 GOSUB 1030
550 REM Zeichen löschen
560 GOSUB 980
```

Programm 13   Teil 3
===========

```
570 NEXT L
580 REM 5) ein Zeichenblock (bestehend
           aus CHR$(250) bis CHR$(253))
           wird zur fließenden Grafik-
           darstellung herangezogen
590 CLS
600 LOCATE 0,0
610 REM Zeichen löschen
620 GOSUB 1230
630 PRINT CHR$(250);CHR$(252)
640 PRINT CHR$(251);CHR$(253)
650 LOCATE 6,10
660 PRINT "SCREEN 1 mit HGR 3"
670 FOR L=1 TO 10
680 REM Zeichen einlesen und darstellen
690 GOSUB 1120
700 REM Zeichen löschen
710 GOSUB 1230
720 NEXT L
730 REM 6) Viele Zeichenblocks (jeweils
           bestehend aus CHR$(250) bis
           CHR$(253)) werden zur flies-
           senden Grafikdarstellung
           herangezogen
740 CLS
750 REM Füllen des Bildschirms mit den
        CHR$-Zeichen 250 bis 253
760 GOSUB 1370
770 LOCATE 6,10
780 PRINT "SCREEN 1 mit HGR 3a"
790 FOR L=1 TO 10
800 REM Zeichen einlesen und darstellen
810 GOSUB 1120
820 REM Zeichen löschen
830 GOSUB 1230
840 NEXT L
850 END
```

Programm 13   Teil 4
===========

```
860 REM U-Programm zum Einlesen von In-
        formationen zum Zeichenaussehen
        von CHR$(250) (Punktdarstellung,
        siehe 1 und 2)
870 RESTORE 1280
880 FOR N=1 TO 20
890 READ A,B
900 VPOKE(250*8+A),B
910 REM Punkt wieder löschen
920 FOR M=1 TO 10
930 NEXT M
940 VPOKE(250*8+A),0
950 NEXT N
960 RETURN
970 REM U-Programm zum Zeichenlöschen
980 FOR N=250*8+0 TO 250*8+7
990 VPOKE N,0
1000 NEXT N
1010 RETURN
1020 REM U-Programm zum Einlesen von In-
         formationen zum Zeichenaussehen
         von CHR$(250) (Zeichnungen in
         CHR$(250), siehe 3 und 4)
1030 RESTORE 1280
1040 FOR N=1 TO 20
1050 READ A,B
1060 B=VPEEK(250*8+A) OR B
1070 VPOKE(250*8+A),B
1080 FOR M=1 TO 10
1090 NEXT M
1100 NEXT N
1110 RETURN
1120 REM U-Programm zum Einlesen von In-
         formationen zu vier CHR$-Zei-
         chen gleichzeitig (Grafik in
         CHR$(250) bis CHR$(253),
         siehe 5 und 6)
```

Programm 13   Teil 5
===========

```
1130 RESTORE 1490
1140 FOR N=1 TO 73
1150 READ A,B,C
1160 C=VPEEK(250*8+A*8+B) OR C
1170 VPOKE (250*8+A*8+B),C
1180 FOR M=1 TO 10
1190 NEXT M
1200 NEXT N
1210 RETURN
1220 REM U-Programm zum Zeichenlöschen
        (CHR$(250) bis CHR$(253))
1230 FOR N=250*8 TO 250*8+32
1240 VPOKE N,0
1250 NEXT N
1260 RETURN
1270 REM DATAs zum Aussehen von
        CHR$(250) in Teile 1 bis 4
1280 DATA 0,128,1,64,2,32,3,16,4,16,5,32
,6,32,7,16,6,8,6,4,6,2,5,1,4,1,3,2,2,4,1
,4,0,8,0,16,0,32,0,64
1290 REM U-Programm zum Füllen des Bild-
        schirms mit CHR$(250)
1300 FORN=0 TO 28
1310 FOR M=0 TO 23
1320 LOCATE N,M
1330 PRINT CHR$(250);
1340 NEXT M,N
1350 RETURN
1360 REM U-Programm zum Füllen des Bild-
        schirms mit Zeichenblocks
        (CHR$(250) bis CHR$(253))
1370 FOR L=1 TO 12
1380 FOR N=0 TO 13
1390 PRINT CHR$(250);CHR$(252);
1400 NEXT N
1410 PRINT
1420 FOR N=0 TO 13
```

Programm 13   Teil 6

```
1430 PRINT CHR$(251);CHR$(253);
1440 NEXT N
1450 PRINT
1460 NEXT L
1470 RETURN
1480 REM DATAs zum Aussehen von
         CHR$(250) bis CHR$(253)
1490 DATA Ø,2,16,Ø,3,16,Ø,4,16,Ø,5,16,Ø,
6,16,Ø,7,16,1,Ø,16,1,1,16,1,2,16,1,3,16,
1,4,32,1,4,64,1,5,128,1,6,64,1,5,32,1,4,
16,1,5,16,1,6,8,1,5,4,1,4,2,1,3,2,1,2,1,
1,1,1,1,Ø,1,Ø,7,1,Ø,6,1,Ø,5,1,Ø,4,1,Ø,3,
1
1500 DATA Ø,2,2,Ø,1,4,Ø,1,8,Ø,1,16,Ø,1,3
2,Ø,1,64,Ø,2,128,Ø,3,128
1510 DATA 2,2,64,2,3,64,2,4,64,2,5,64,2,
6,64,2,7,64,3,Ø,64,3,1,64,3,2,64,3,3,64,
3,4,64,3,5,64,3,6,64,2,2,128,2,1,64,2,1,
32,2,1,16,2,1,8,2,1,4,2,2,2,2,3,1,2,4,1,
2,5,2,2,6,4,2,7,8,3,Ø,4,3,1,2,3,2,1
1520 DATA 3,3,1,3,4,2,3,5,4,3,5,8,3,6,16
,3,6,32,3,6,64,3,6,128
```

Kurzbeschreibung zu Programm 14

Dieses Programm ermöglicht es Ihnen, über die VPEEK- und VPOKE-Befehle hinwegzusehen und anstattdessen sehr luxuriös den Zeichensatz Ihres MSX-Computers zu verändern.

Bei der Programmplanung tauchte allerdings zu Beginn eine Problematik auf: Wenn Sie alle Zeichen des Zeichensatzes verändern können (z.B. auch für den Buchtaben 'A' ein 'B' oder gar für die '1' eine '2' abspeichern können), würden sich entsrechende Hilfstexte oder Zahlangaben ebenfalls ändern. Also wurde ein Zeichensatz bzw. eine Bildsprache entwickelt, die aus nur 3 Zeichen besteht: Leerzeichen, vertikaler Balken und Punkt. Diese 3 Zeichen können Sie mit Programm 14 nicht verändern. Wie arbeitet nun diese Zeichensprache z.B. zur Wiedergabe einer Zahl?

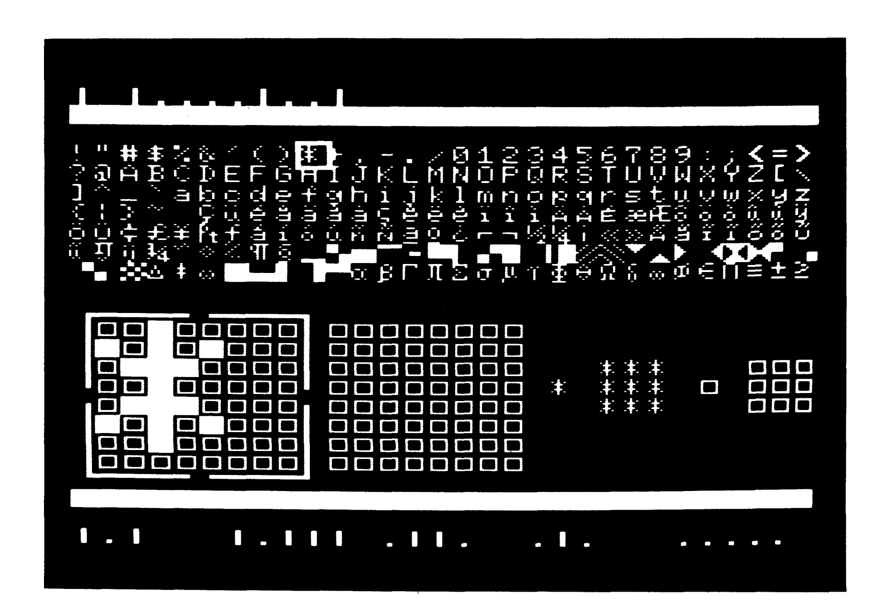
Der Punkt dient zur Anzeige der Einer, Zehner und Hunderter, der Balken trennt die Einer von den Zehnern und diese wiederum von den Hundertern. Das Leerzeichen zeigt eine leere Menge an. Die Zahl 214 sähe dann so aus:

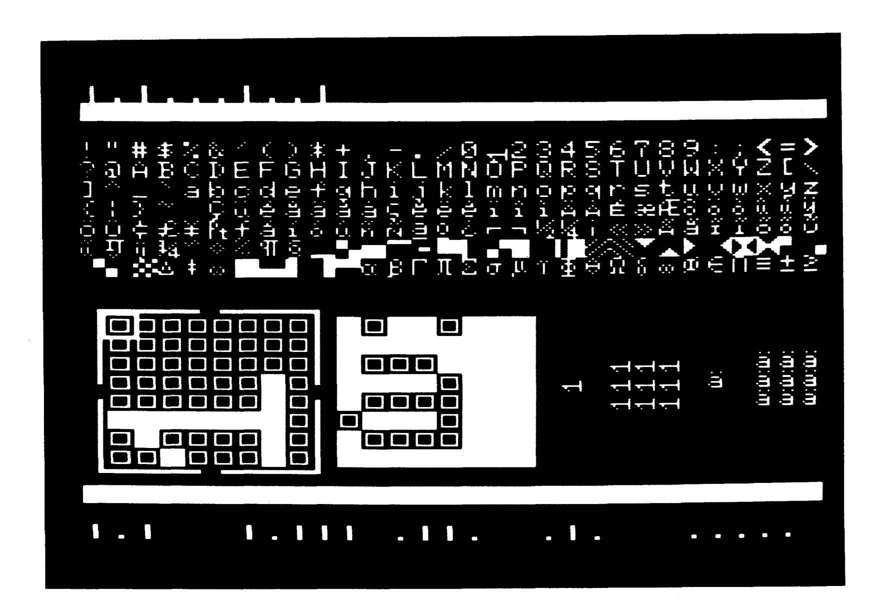
2 Hunderter = 2 Punkte. Ein Balken zur Abtrennung.
1 Zehner = 1 Punkt. Ein Balken zur Abtrennung.
4 Einer = 4 Punkte. Ein Balken zur Schlußabtrennung.

Die Funktionstasten haben folgende Bedeutung:

F1/F6: Balken, Punkt, Balken: Zeichensatz laden/speichern
F2/F7: drei Balken, ein Punkt, ein Balken: Zeichenblock 1/2
F3: Punkt, zwei Balken, Punkt: Spiegelung
F4: Punkt, Balken, Punkt: Drehung nach rechts
F5: fünf Punkte: Help an/aus
F8: Balken, Leerzeichen, Balken: löschen
F9: Punkt, Balken, Leerzeichen, Balken: inverse Darstell.
F10: Leerzeichen, Balken, Leerzeichen, 2 Balken: Wechsel

Mit der Tabulatortaste gehen Sie in den Editor (mit Leertaste Punkte löschen und setzen), mit ENTER geben Sie Ihr OK zur Zeichenänderung laut Ihrer Eingabe und mit ESC kehren Sie in den Zeichensatz zurück.





Programm 14 Teil 1

```
10 REM Programm 14
20 REM MSX Grafik und Sound
30 REM Copyright 1985 DATA BECKER &
                     Rainer Lüers
40 REM Charactergenerator, der es Ihnen
       leicht macht, eigene Zeichensätze
       zu kreieren bzw. den bestehenden
       MSX-Zeichensatz in Teilen zu ver-
       ändern. Zehn verschiedene Funkti-
50 REM onen wie Inversdarstellung, Spie-
       gelung usw. erleichtern Ihnen
       diese Arbeit erheblich.
60 REM Der Arbeitsbildschirm beginnt an
       VRAM-Speicherstelle 6144
70 VDP(2)=6
80 REM Der Arbeitszeichensatz beginnt
       an VRAM-Speicherstelle 0
90 VDP(4)=0
100 DEFINT A-Z
110 CLEAR 2000
120 DIM H$(13)
130 REM kein Programmabbruch mit unauf-
        geräumtem Bildschirm möglich
140 ON STOP GOSUB 5530
150 STOP ON
160 REM Start der Unterprogramme für
        verschiedene Funktionen
170 ON KEY GOSUB 3600,3710,3800,4180,456
0,4590,4700,4790,5070,5350
180 FOR N=1 TO 10
190 KEY(N) ON
200 NEXT N
210 REM Definition der CHR$-Zeichen
        251 bis 254 u.a. für Funktions-
        tastenanzeige und Zahlerstel-
        lung
220 DATA 00000000
```

Programm 14   Teil 2
===========

```
230 DATA Ø111111Ø
240 DATA Ø1ØØØØ1Ø
250 DATA Ø1ØØØØ1Ø
260 DATA Ø1ØØØØ1Ø
270 DATA Ø1ØØØØ1Ø
280 DATA Ø111111Ø
290 DATA ØØØØØØØØ
300 '
310 DATA 11111111
320 DATA 11111111
330 DATA 11111111
340 DATA 11111111
350 DATA 11111111
360 DATA 11111111
370 DATA 11111111
380 DATA 11111111
390 '
400 DATA ØØØØØØØØ
410 DATA ØØØØØØØØ
420 DATA ØØØØØØØØ
430 DATA ØØØØØØØØ
440 DATA ØØØØØØØØ
450 DATA ØØØØØØØØ
460 DATA ØØØ11ØØØ
470 DATA ØØØ11ØØØ
480 '
490 DATA ØØØØØØØØ
500 DATA ØØØ11ØØØ
510 DATA ØØØ11ØØØ
520 DATA ØØØ11ØØØ
530 DATA ØØØ11ØØØ
540 DATA ØØØ11ØØØ
550 DATA ØØØ11ØØØ
560 DATA ØØØ11ØØØ
570 '
580 REM Aussehen von vier 16*16-Sprites
       für Blockumrandung
```

Programm 14   Teil 3
===========

```
590 DATA 1111111111111111
600 DATA 1000000000000000
610 DATA 1000000000000000
620 DATA 1000000000000000
630 DATA 1000000000000000
640 DATA 1000000000000000
650 DATA 1000000000000000
660 DATA 1000000000000000
670 DATA 1000000000000000
680 DATA 1000000000000000
690 DATA 1000000000000000
700 DATA 1000000000000000
710 DATA 1000000000000000
720 DATA 1000000000000000
730 DATA 1000000000000000
740 DATA 1000000000000000
750 '
760 DATA 1111111111111111
770 DATA 0000000000000001
780 DATA 0000000000000001
790 DATA 0000000000000001
800 DATA 0000000000000001
810 DATA 0000000000000001
820 DATA 0000000000000001
830 DATA 0000000000000001
840 DATA 0000000000000001
850 DATA 0000000000000001
860 DATA 0000000000000001
870 DATA 0000000000000001
880 DATA 0000000000000001
890 DATA 0000000000000001
900 DATA 0000000000000001
910 DATA 0000000000000001
920 '
930 DATA 1000000000000000
940 DATA 1000000000000000
950 DATA 1000000000000000
```

Programm 14 Teil 4

```
960 DATA 1000000000000000
970 DATA 1000000000000000
980 DATA 1000000000000000
990 DATA 1000000000000000
1000 DATA 1000000000000000
1010 DATA 1000000000000000
1020 DATA 1000000000000000
1030 DATA 1000000000000000
1040 DATA 1000000000000000
1050 DATA 1000000000000000
1060 DATA 1000000000000000
1070 DATA 1000000000000000
1080 DATA 1111111111111111
1090 '
1100 DATA 0000000000000001
1110 DATA 0000000000000001
1120 DATA 0000000000000001
1130 DATA 0000000000000001
1140 DATA 0000000000000001
1150 DATA 0000000000000001
1160 DATA 0000000000000001
1170 DATA 0000000000000001
1180 DATA 0000000000000001
1190 DATA 0000000000000001
1200 DATA 0000000000000001
1210 DATA 0000000000000001
1220 DATA 0000000000000001
1230 DATA 0000000000000001
1240 DATA 0000000000000001
1250 DATA 1111111111111111
1260 '
1270 REM Sprite zur CHR$-Auswahl und
         Editierung
1280 DATA 1111110000000000
1290 DATA 1000010000000000
1300 DATA 1000010000000000
1310 DATA 1000010000000000
```

Programm 14   Teil 5

```
1320 DATA 1000010000000000
1330 DATA 1111110000000000
1340 DATA 0000000000000000
1350 DATA 0000000000000000
1360 DATA 0000000000000000
1370 DATA 0000000000000000
1380 DATA 0000000000000000
1390 DATA 0000000000000000
1400 DATA 0000000000000000
1410 DATA 0000000000000000
1420 DATA 0000000000000000
1430 DATA 0000000000000000
1440 '
1450 REM Codierung der zehn
         Funktionstasten
1460 KEY 1,CHR$(254)+CHR$(253)+CHR$(254)
1470 KEY 2,CHR$(254)+CHR$(253)+CHR$(254)
+CHR$(254)+CHR$(254)
1480 KEY 3,CHR$(253)+CHR$(254)+CHR$(254)
+CHR$(253)
1490 KEY 4,CHR$(253)+CHR$(254)+CHR$(253)
1500 KEY 5,STRING$(5,CHR$(253))
1510 KEY 6,CHR$(254)+CHR$(253)+CHR$(253)
+CHR$(254)
1520 KEY 7,CHR$(254)+CHR$(254)+CHR$(254)
+CHR$(253)+CHR$(254)
1530 KEY 8,CHR$(254)+" "+CHR$(254)
1540 KEY 9,CHR$(253)+CHR$(254)+" "+CHR$(
254)
1550 KEY 10," "+CHR$(254)+" "+CHR$(254)+
CHR$(254)
1560 REM Anwahl von SCREEN 1 mit ver-
         größerter 16*16-Spritedarstel-
         lung
1570 SCREEN 1,3
1580 COLOR 15,1
1590 WIDTH 30
```

Programm 14   Teil 6

```
1600 KEY ON
1610 PRINT "Bitte ein bisschen Geduld ..
."
1620 REM Einladen des Charactercodes
         in CHR$-Zeichen 251 bis 254
1630 FOR N=Ø TO 3
1640 FOR M=Ø TO 7
1650 READ A$
1660 A=VAL("&b"+A$)
1670 VPOKE (BASE(7)+((251+N)*8))+M,A
1680 NEXT M
1690 NEXT N
17ØØ REM Kopieren des Zeichensatzes in
         die VRAM-Speicherstellen
         1Ø24Ø ff. (VDP(4)=5))
171Ø FOR N%=Ø TO 2Ø48
172Ø VPOKE N%+1Ø24Ø,VPEEK(N%)
173Ø NEXT N%
174Ø REM Abspeicherung eines zweiten
         Bildschirms als Helpbildschirm
         (VDP(2)=9)
175Ø H$(Ø)="     Funktionstastenbelegung"
176Ø H$(1)="     "+STRING$(23,219)
177Ø H$(2)=STRING$(32," ")
178Ø H$(3)=STRING$(32," ")
179Ø H$(4)="     1 = Zeichensatz laden"
18ØØ H$(5)="     2 = Zeichenblock 1"
181Ø H$(6)="     3 = Spiegelung"
182Ø H$(7)="     4 = Drehung nach rechts"
183Ø H$(8)="     8 = Help an/aus"
184Ø H$(9)="     6 = Zeichensatz abspeich
ern"
185Ø H$(1Ø)="     7 = Zeichenblock 2"
186Ø H$(11)="     8 = loeschen"
187Ø H$(12)="     9 = inverse Darstellung
"
188Ø H$(13)="    1Ø = vertauschen 1 - 2"
```

Programm 14   Teil 7
===========

```
1890 REM Löschen des zweiten Bildschirms
1900 FOR N=9216 TO 9216+767
1910 VPOKE N,32
1920 NEXT N
1930 REM Abbildung der Helpzeilen
1940 FOR N=0 TO 13
1950 FOR M=1 TO LEN(H$(N))
1960 VPOKE 9215+(N*32+M),ASC(MID$(H$(N),
M,1))
1970 NEXT M
1980 NEXT N
1990 REM Einladen der 5 Sprites aus
         den DATA-Zeilen
2000 FOR N=1 TO 5
2010 S$=SPACE$(32)
2020 FOR M=1 TO 16
2030 READ A$
2040 MID$(S$,M,1)=CHR$(VAL("&b"+LEFT$(A$
,8)))
2050 MID$(S$,M+16,1)=CHR$(VAL("&b"+RIGHT
$(A$,8)))
2060 NEXT M
2070 SPRITE$(N)=S$
2080 NEXT N
2090 REM sichtbarer Programmstart
2100 CLS
2110 PRINT
2120 PRINT STRING$(30,252);
2130 PRINT
2140 REM Abbildung der CHR$-Zeichen
         33 bis 126
2150 FOR N=33 TO 126
2160 PRINT CHR$(N);
2170 NEXT N
2180 PRINT " ";
2190 REM Abbildung der CHR$-Zeichen
         128 bs 242
```

Programm 14   Teil 8
===========

```
2200 FOR N=128 TO 242
2210 PRINT CHR$(N);
2220 NEXT N
2230 PRINT
2240 PRINT
2250 REM Abbildung der 2 Editierblocks
2260 FOR N=1 TO 8
2270 PRINT " ";STRING$(8,CHR$(251));" ";
STRING$(8,CHR$(251))
2280 NEXT N
2290 REM Abbildung von jeweils 10 Zei-
        chen, die als Kontrolle der
        Editierung gelten
2300 LOCATE 19,15
2310 PRINT CHR$(251);
2320 LOCATE 25,15
2330 PRINT CHR$(251);
2340 FOR N=21 TO 23
2350 FOR M=14 TO 16
2360 LOCATE N,M
2370 PRINT CHR$(251);
2380 NEXT M,N
2390 FOR N=27 TO 29
2400 FOR M=14 TO 16
2410 LOCATE N,M
2420 PRINT CHR$(251);
2430 NEXT M,N
2440 LOCATE 0,21
2450 PRINT STRING$(30,CHR$(252));
2460 REM Einzeichnen der 4 Blocksprites
        und des Editiersprites
2470 PUT SPRITE 5,(6,21),15
2480 PUT SPRITE 1,(13,92),15
2490 PUT SPRITE 2,(51,92),15
2500 PUT SPRITE 3,(13,130),15
2510 PUT SPRITE 4,(51,130),15
2520 REM Ausgangspunkt des Character-
```

Programm 14   Teil 9
===========

```
        cursorsprites
2530 Z1=6
2540 Z2=21
2550 C1=33
2560 REM Tastaturabfrage-Kontrolle
2570 A$=INKEY$
2580 IF A$="" THEN GOTO 2570
2590 REM ESC -> Wechsel von CHR$-Anzeige
        zum Blockeditor
2600 IF ASC(A$)=27 THEN ED=1:GOTO 3210
2610 REM Pfeiltastenabfrage
2620 IF ASC(A$)=28 THEN IF (Z1<>238) THE
N  Z1=Z1+8:PUT SPRITE 5,(Z1,Z2),15:C1=C1
+1:GOTO 3120
2630 IF ASC(A$)=29 AND (Z1<>6) THEN  Z1=
Z1-8:PUT SPRITE 5,(Z1,Z2),15:C1=C1-1:GOT
O 3120
2640 IF ASC(A$)=30 AND (Z2<>21) THEN Z2=
Z2-8:PUT SPRITE 5,(Z1,Z2),15:C1=C1-30:GO
TO 3120
2650 IF ASC(A$)=31 AND (Z2<>69) THEN Z2=
Z2+8:PUT SPRITE 5,(Z1,Z2),15:C1=C1+30:GO
TO 3120
2660 REM TAB -> angesteuertes Zeichen im
        aktuellen Block vergrößert dar-
        stellen
2670 IF ASC(A$)=9 THEN IF M1=2 THEN GOTO
 2900 ELSE GOTO 2690 ELSE GOTO 2570
2680 REM linker Block (M1=1)
2690 FOR N=0 TO 7
2700 A$(N)=BIN$(VPEEK(BASE(7)+(C1*8)+N))
2710 A$(N)=STRING$(8-LEN(A$(N)),"0")+A$(
N)
2720 NEXT N
2730 FOR N=0 TO 7
2740 FOR M=0 TO 7
2750 LOCATE M+1,N+12
```

Programm 14   Teil 1Ø
===========

```
2760 IF MID$(A$(N),M+1,1)="1" THEN PRINT
 CHR$(252); ELSE PRINT CHR$(251);
2770 A(M+1,N+1)=VAL(MID$(A$(N),M+1,1))
2780 NEXT M
2790 NEXT N
2800 LOCATE 19,15
2810 PRINT CHR$(C1);
2820 FOR N=21 TO 23
2830 FOR M=14 TO 16
2840 LOCATE N,M
2850 PRINT CHR$(C1);
2860 NEXT M
2870 NEXT N
2880 GOTO 2570
2890 REM rechter Block (M1=2)
2900 FOR N=Ø TO 7
2910 B$(N)=BIN$(VPEEK(BASE(7)+(C1*8)+N))
2920 B$(N)=STRING$(8-LEN(B$(N)),"Ø")+B$(
N)
2930 NEXT N
2940 FOR N=Ø TO 7
2950 FOR M=Ø TO 7
2960 LOCATE M+1Ø,N+12
2970 IF MID$(B$(N),M+1,1)="1" THEN PRINT
 CHR$(252); ELSE PRINT CHR$(251);
2980 B(M+1,N+1)=VAL(MID$(B$(N),M+1,1))
2990 NEXT M
3000 NEXT N
3010 LOCATE 25,15
3020 PRINT CHR$(C1);
3030 FOR N=27 TO 29
3040 FOR M=14 TO 16
3050 LOCATE N,M
3060 PRINT CHR$(C1);
3070 NEXT M
3080 NEXT N
3090 REM Anzeige der CHR$-Zeichennummer
```

Programm 14   Teil 11
===========

mit Punkten und Strichen
(CHR$(253) und CHR$(254))

```
3100 GOTO 2570
3110 NEXT N
3120 LOCATE 0,0
3130 PRINT SPACE$(23);
3140 LOCATE 0,0
3150 A$=CHR$(254)
3160 IF C1>99 THEN Z=INT(C1/100):A$=A$+S
TRING$(Z,CHR$(253))+CHR$(254):C2=C1-(Z*1
00) ELSE A$=A$+" "+CHR$(254):C2=C1
3170 IF C2>9 THEN Z=INT(C2/10):A$=A$+STR
ING$(Z,CHR$(253))+CHR$(254):C3=C2-(Z*10)
 ELSE A$=A$+" "+CHR$(254):C3=C2
3180 IF C3>0 THEN Z=C3:A$=A$+STRING$(Z,C
HR$(253))+CHR$(254) ELSE A$=A$+" "+CHR$(
254)
3190 PRINT A$;
3200 GOTO 2570
3210 REM Arbeit als Editor mit ver-
        größerter CHR$-Darstellung
3220 IF M1=2 THEN GOTO 3420
3230 REM Arbeit im linken Block (M1=1)
3240 M5=14
3250 M6=93
3260 M7=1
3270 M8=1
3280 PUT SPRITE 5,(M5,M6),15
3290 A$=INKEY$:IF A$="" THEN GOTO 3290
3300 REM ENTER -> übernahme der Editier-
        funktion in das CHR$-Zeichen
3310 IF ASC(A$)=13 THEN FOR N=0 TO 7:FOR
 M=1 TO 8:Q$=Q$+RIGHT$(STR$(A(M,N+1)),1)
:NEXT M:VPOKE C1*8+N,VAL("&b"+Q$):Q$="":
NEXT N
3320 REM Leertaste -> Pixel setzen
        bzw. löschen
```

Programm 14   Teil 12
===========

```
3330 IF A$=" " THEN LOCATE 0+M7,11+M8:IF
 A(M7,M8)=1 THEN PRINT CHR$(251);:A(M7,M
8)=0 ELSE PRINT CHR$(252);:A(M7,M8)=1
3340 REM ESC -> Wechsel vom Blockeditor
         zu CHR$-Anzeige
3350 IF ASC(A$)=27 THEN ED=0:PUT SPRITE
5,(Z1,Z2),15:GOTO 2570
3360 REM Pfeiltastenabfrage
3370 IF ASC(A$)=28 AND M7<>8 THEN M5=M5+
8:M7=M7+1:PUT SPRITE 5,(M5,M6),15:GOTO 3
290
3380 IF ASC(A$)=29 AND M7<>1 THEN M5=M5-
8:M7=M7-1:PUT SPRITE 5,(M5,M6),15:GOTO 3
290
3390 IF ASC(A$)=31 AND M8<>8 THEN M6=M6+
8:M8=M8+1:PUT SPRITE 5,(M5,M6),15:GOTO 3
290
3400 IF ASC(A$)=30 AND M8<>1 THEN M6=M6-
8:M8=M8-1:PUT SPRITE 5,(M5,M6),15:GOTO 3
290 ELSE GOTO 3290
3410 REM Arbeit im rechten Block (M1=2)
3420 M5=86
3430 M6=93
3440 M7=1
3450 M8=1
3460 PUT SPRITE 5,(M5,M6),15
3470 A$=INKEY$
3480 IF A$="" THEN GOTO 3470
3490 REM ENTER -> übernahme der Editier-
         funktion in das CHR$-Zeichen
3500 IF ASC(A$)=13 THEN FOR N=0 TO 7:FOR
 M=1 TO 8:Q$=Q$+RIGHT$(STR$(B(M,N+1)),1)
:NEXT M:VPOKE C1*8+N,VAL("&b"+Q$):Q$="":
NEXT N
3510 REM Leertaste -> Pixel setzen
         bzw. löschen
3520 IF A$=" " THEN LOCATE 9+M7,11+M8:IF
```

Programm 14   Teil 13
===========

```
 B(M7,M8)=1 THEN PRINT CHR$(251);:B(M7,M
8)=Ø ELSE PRINT CHR$(252);:B(M7,M8)=1
353Ø REM ESC -> Wechsel vom Blockeditor
         zu CHR$-Anzeige
354Ø IF ASC(A$)=27 THEN ED=Ø:PUT SPRITE
5,(Z1,Z2),15:GOTO 257Ø
355Ø REM Pfeiltastenabfrage
356Ø IF ASC(A$)=28 AND M7<>8 THEN M5=M5+
8:M7=M7+1:PUT SPRITE 5,(M5,M6),15:GOTO 3
47Ø
357Ø IF ASC(A$)=29 AND M7<>1 THEN M5=M5-
8:M7=M7-1:PUT SPRITE 5,(M5,M6),15:GOTO 3
47Ø
358Ø IF ASC(A$)=31 AND M8<>8 THEN M6=M6+
8:M8=M8+1:PUT SPRITE 5,(M5,M6),15:GOTO 3
47Ø
359Ø IF ASC(A$)=3Ø AND M8<>1 THEN M6=M6-
8:M8=M8-1:PUT SPRITE 5,(M5,M6),15:GOTO 3
47Ø ELSE GOTO 347Ø
36ØØ REM Funktionstaste 1
         Zeichensatz laden
361Ø LOCATE 1Ø,Ø
362Ø OPEN "charac" FOR INPUT AS #1
363Ø FOR N=33 TO 249
364Ø FOR M=Ø TO 7
365Ø INPUT #1,A%
366Ø VPOKE(BASE(7)+(N*8)+M),A%
367Ø NEXT M
368Ø NEXT N
369Ø CLOSE
37ØØ RETURN
371Ø REM Funktionstaste 2
         linken Block anspringen
372Ø REM nicht möglich, wenn Sie sich
         gerade in einem Editierblock
         aufhalten (ED=1)
373Ø IF ED=1 THEN RETURN
```

Programm 14   Teil 14
===========

```
3740 PUT SPRITE 1,(13,92),15
3750 PUT SPRITE 2,(51,92),15
3760 PUT SPRITE 3,(13,130),15
3770 PUT SPRITE 4,(51,130),15
3780 M1=1
3790 RETURN
3800 REM Funktionstaste 3
         Spiegelung von Zeichen
3810 IF M1=2 THEN GOTO 4010
3820 REM Arbeit im linken Block (M1=1)
3830 FOR N=1 TO 8
3840 FOR M=1 TO 8
3850 C(N,M)=A(ABS(N-9),M)
3860 NEXT M
3870 NEXT N
3880 FOR N=1 TO 8
3890 FOR M=1 TO 8
3900 A(N,M)=C(N,M)
3910 NEXT M
3920 NEXT N
3930 FOR N=0 TO 7
3940 FOR M=0 TO 7
3950 LOCATE N+1,M+12
3960 IF A(N+1,M+1)=1 THEN PRINT CHR$(252
);ELSE PRINT CHR$(251);
3970 NEXT M
3980 NEXT N
3990 RETURN
4000 REM Arbeit im rechten Block (M1=2)
4010 FOR N=1 TO 8
4020 FOR M=1 TO 8
4030 C(N,M)=B(ABS(N-9),M)
4040 NEXT M
4050 NEXT N
4060 FOR N=1 TO 8
4070 FOR M=1 TO 8
4080 B(N,M)=C(N,M)
```

Programm 14   Teil 15
===========

```
4090 NEXT M
4100 NEXT N
4110 FOR N=0 TO 7
4120 FOR M=0 TO 7
4130 LOCATE N+10,M+12
4140 IF B(N+1,M+1)=1 THEN PRINT CHR$(252
);ELSE PRINT CHR$(251);
4150 NEXT M
4160 NEXT N
4170 RETURN
4180 REM Funktionstaste 4
        Drehung von Zeichen
4190 IF M1=2 THEN GOTO 4390
4200 REM Arbeit im linken Block
4210 FOR N=1 TO 8
4220 FOR M=1 TO 8
4230 C(N,M)=A(M,9-N)
4240 NEXT M
4250 NEXT N
4260 FOR N=1 TO 8
4270 FOR M=1 TO 8
4280 A(N,M)=C(N,M)
4290 NEXT M
4300 NEXT N
4310 FOR N=0 TO 7
4320 FOR M=0 TO 7
4330 LOCATE N+1,M+12
4340 IF A(N+1,M+1)=1 THEN PRINT CHR$(252
); ELSE PRINT CHR$(251);
4350 NEXT M
4360 NEXT N
4370 RETURN
4380 REM Arbeit im rechten Block (M1=2)
4390 FOR N=1 TO 8
4400 FOR M=1 TO 8
4410 C(N,M)=B(M,9-N)
4420 NEXT M
```

Programm 14   Teil 16
===========

```
4430 NEXT N
4440 FOR N=1 TO 8
4450 FOR M=1 TO 8
4460 B(N,M)=C(N,M)
4470 NEXT M
4480 NEXT N
4490 FOR N=0 TO 7
4500 FOR M=0 TO 7
4510 LOCATE N+10,M+12
4520 IF B(N+1,M+1)=1 THEN PRINT CHR$(252
); ELSE PRINT CHR$(251);
4530 NEXT M
4540 NEXT N
4550 RETURN
4560 REM Funktionstaste 5
         Helpbildschirm an-/ausstellen
         (Wechsel zwischen Bildschirmen
         (VDP(2)) und Zeichensätzen
         (VDP(4))
4570 IF VDP(2)=6 THEN FOR N=6919 TO 6935
 STEP 4:VPOKE N,0:NEXT N:VDP(4)=5:VDP(2)
=9 ELSE VDP(4)=0:VDP(2)=6:FOR N=6919 TO
6935 STEP 4:VPOKE N,15:NEXT N
4580 RETURN
4590 REM Funktionstaste 6
         Zeichensatz abspeichern
4600 LOCATE 10,0
4610 OPEN "charac" FOR OUTPUT AS #1
4620 FOR N=33 TO 249
4630 FOR M=0 TO 7
4640 A%=VPEEK(BASE(7)+(N*8)+M)
4650 PRINT #1,A%
4660 NEXT M
4670 NEXT N
4680 CLOSE
4690 RETURN
4700 REM Funktionstaste 7
```

Programm 14   Teil 17
===========

```
          rechten Block anspringen
4710 REM nicht möglich, wenn Sie sich in
          einem Editiertblock aufhalten
          (ED=1)
4720 IF ED=1 THEN RETURN
4730 PUT SPRITE 1,(85,92),15
4740 PUT SPRITE 2,(123,92),15
4750 PUT SPRITE 3,(85,130),15
4760 PUT SPRITE 4,(123,130),15
4770 M1=2
4780 RETURN
4790 REM Funktionstaste 8
          Löschen von Zeichen
4800 IF M1=2 THEN GOTO 4950
4810 REM Arbeit im linken Block (M1=1)
4820 FOR N=1 TO 8
4830 FOR M=1 TO 8
4840 A(N,M)=0
4850 NEXT M
4860 NEXT N
4870 FOR N=0 TO 7
4880 FOR M=0 TO 7
4890 LOCATE N+1,M+12
4900 PRINT CHR$(251);
4910 NEXT M
4920 NEXT N
4930 RETURN
4940 REM Arbeit im rechten Block (M1=2)
4950 FOR N=1 TO 8
4960 FOR M=1 TO 8
4970 B(N,M)=0
4980 NEXT M
4990 NEXT N
5000 FOR N=0 TO 7
5010 FOR M=0 TO 7
5020 LOCATE N+10,M+12
5030 PRINT CHR$(251);
```

Programm 14   Teil 18
===========

```
5Ø4Ø NEXT M
5Ø5Ø NEXT N
5Ø6Ø RETURN
5Ø7Ø REM Funktionstaste 9
         Invertieren von Zeichen
5Ø8Ø IF M1=2 THEN GOTO 523Ø
5Ø9Ø REM Arbeit im linken Block (M1=1)
51ØØ FOR N=1 TO 8
511Ø FOR M=1 TO 8
512Ø IF A(N,M)=1 THEN A(N,M)=Ø ELSE A(N,
M)=1
513Ø NEXT M
514Ø NEXT N
515Ø FOR N=Ø TO 7
516Ø FOR M=Ø TO 7
517Ø LOCATE N+1,M+12
518Ø IF A(N+1,M+1)=Ø THEN PRINT CHR$(251
); ELSE PRINT CHR$(252);
519Ø NEXT M
52ØØ NEXT N
521Ø RETURN
522Ø REM Arbeit im rechten Block (M1=2)
523Ø FOR N=1 TO 8
524Ø FOR M=1 TO 8
525Ø IF B(N,M)=1 THEN B(N,M)=Ø ELSE B(N,
M)=1
526Ø NEXT M
527Ø NEXT N
528Ø FOR N=Ø TO 7
529Ø FOR M=Ø TO 7
53ØØ LOCATE N+1Ø,M+12
531Ø IF B(N+1,M+1)=Ø THEN PRINT CHR$(251
); ELSE PRINT CHR$(252);
532Ø NEXT M
533Ø NEXT N
534Ø RETURN
535Ø REM Funktionstaste 1Ø
```

Programm 14   Teil 19
===========

Austauschen der Inhalte von
Block 1 und Block 2

```
5360 FOR N=1 TO 8
5370 FOR M=1 TO 8
5380 C(N,M)=A(N,M)
5390 A(N,M)=B(N,M)
5400 B(N,M)=C(N,M)
5410 NEXT M
5420 NEXT N
5430 FOR N=0 TO 7
5440 FOR M=0 TO 7
5450 LOCATE N+1,M+12
5460 IF A(N+1,M+1)=1 THEN PRINT CHR$(252
); ELSE PRINT CHR$(251);
5470 LOCATE N+10,M+12
5480 IF B(N+1,M+1)=1 THEN PRINT CHR$(252
); ELSE PRINT CHR$(251);
5490 NEXT M
5500 NEXT N
5510 RETURN
5520 REM bei Programmabbruch wird alles
        sauber beendet
5530 VDP(2)=6
5540 VDP(4)=0
5550 CLS
5560 SCREEN,3
5570 END
```

Kurzbeschreibung zu Programm FRETY

Guten Tag, lieber Leser!

Das Spiel, dessen Listing Sie hier sehen, handelt von einem kleinen Affen, der aus dem Zoo ausgebrochen ist und nun vor den Wärtern fliehen muß.

Dabei muß er zuerst so schnell wie möglich rennen, da ihm schon ein Verfolger auf den Fersen ist.

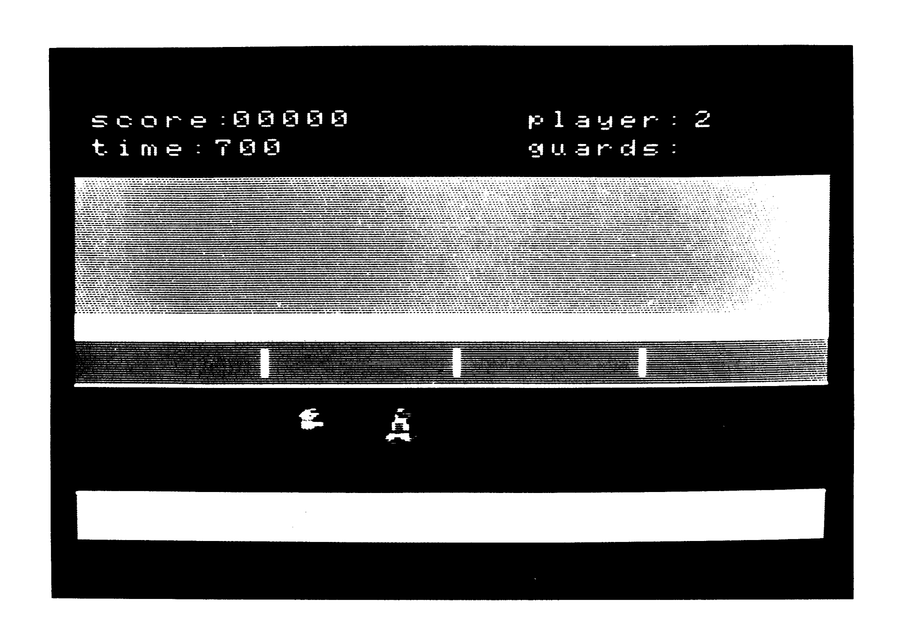
Ist man am Ende der Straße angekommen, so entsteht das zweite Bild, wo es darauf ankommt, Bananen zu sammeln und Wärter damit zu bewerfen.

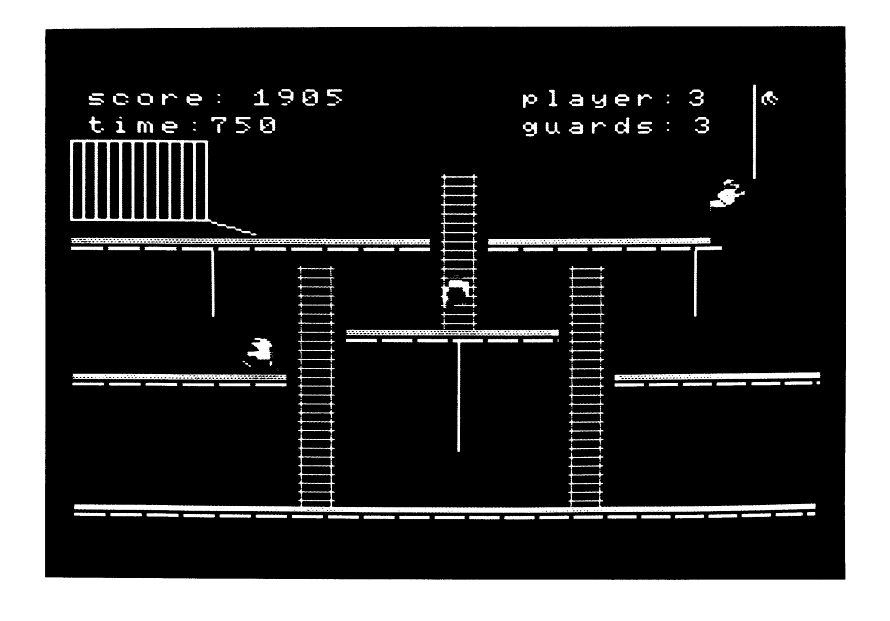
Wenn die Anzeige "guards:" Null anzeigt, kommt wieder das erste Bild, wo man rennen muß, doch der Verfolger ist schneller geworden.

Bei jedem weiteren Durchlauf muß man mehr Wärter abwerfen, so daß es nachher kritisch werden kann, wenn die Zeit abgelaufen ist!

Viel Spaß beim Spielen

Olaf Otis
(Künstlername)


score:00000
time:700
player:2
guards:


score: 1905
time:750
player:3
guards: 3

FRETY Teil 1

```
10 REM FRETY
20 REM MSX Grafik und Sound
30 REM Copyright 1985 DATA BECKER &
       Rainer Lüers & Olaf Otis
40 DEFINT C-Y,A
50 SCREEN,,0
60 OPEN "grp:"FOR OUTPUT AS #1
70 DIM M(20)
80 DIM SD(8)
90 COLOR 10,1,1
100 SCREEN 2
110 LINE(30,40)-(220,130),15,B
120 DRAW "bm 110,85"
130 PRINT #1,"FRETY"
140 DRAW "bm 28,150"
150 PRINT #1,"WOLLEN SIE DIE ANLEITUNG"
160 DRAW "bm 78,158"
170 PRINT #1,"LESEN?(J/N)"
180 A$=INKEY$
190 IF A$="J" OR A$="j" THEN GOSUB 6920
ELSE IF A$="N" OR A$="n" THEN GOTO 210
200 GOTO 180
210 SCREEN 2
220 DRAW "bm 8,10"
230 PRINT #1,"WOLLEN SIE MIT DER TASTATU
R(0)"
240 DRAW "bm 8,20"
250 PRINT #1,"ODER MIT DEM JOYSTICK (1)"
260 DRAW "bm 74,30"
270 PRINT #1,"SPIELEN ?"
280 DRAW "bm 2,6"
290 A$=INKEY$
300 IF A$="1" THEN AZ=1:GOTO 330
310 IF A$="0" THEN AZ=0:GOTO 330
320 GOTO 290
330 SCREEN 2,2
340 FOR A=1 TO 30 STEP 8
```

FRETY Teil 2
=====

```
350 LINE(30,40)-(220,130),15,B
360 DRAW "bm 48,80"
370 COLOR 10
380 PRINT #1,"ES DAUERT NOCH ETWAS"
390 DRAW "bm 70,90"
400 PRINT #1,"BIS ZUM START!"
410 COLOR 14
420 FOR L=1 TO 20
430 READ M(L)
440 NEXT L
450 DATA 1,4,7,9,11,12,13,14,15,15,15,15
,14,13,12,11,9,7,4,0
460 FOR CB=1 TO 8
470 READ SD(CB)
480 NEXT CB
490 DATA 5,5,4,4,3,3,4,4
500 CB=0
510 ' Spritedefinition
520 S$=SPACE$(32)
530 FOR Q=0 TO 13
540 FOR I=1 TO 16
550 READ A$
560 MID$(S$,I,1)=CHR$(VAL("&B"+LEFT$(A$,
8)))
570 MID$(S$,I+16)=CHR$(VAL("&B"+RIGHT$(A
$,8)))
580 NEXT I
590 SPRITE$(Q)=S$
600 NEXT Q
610 RESTORE 1920
620 FOR T=14 TO 25
630 FOR N=1 TO 16
640 READ A$
650 B$=SPACE$(16)
660 FOR U=1 TO LEN(A$)
670 MID$(B$,LEN(A$)+1-U,1)=MID$(A$,U,1)
680 NEXT U
```

FRETY Teil 3
=====

```
690 A$=B$
700 MID$(S$,N,1)=CHR$(VAL("&B"+LEFT$(A$,
8)))
710 MID$(S$,N+16)=CHR$(VAL("&B"+RIGHT$(A
$,8)))
720 NEXT N
730 SPRITE$(T)=S$
740 NEXT T
750 ' Anzeige
760 RT=0
770 CV=4
780 ZX=0
790 FA=2
800 ZE=1
810 F$="1"
820 DRAW "bm 52,2"
830 PRINT #1,"00000"
840 DRAW "bm 208,2"
850 PRINT #1,"4"
860 DRAW "bm  6,2"
870 PRINT #1,"score:                player:"
880 COLOR 1
890 DRAW "bm 80,14"
900 PRINT #1,"    "
910 COLOR 14
920 DRAW "bm  6 ,14"
930 PRINT #1,"time:700              guards:"
940 COLOR 1
950 DRAW "bm 204,14"
960 PRINT #1,"  "
970 COLOR 14
980 DRAW "bm 202,14"
990 PRINT #1,FA
1000 IF F$="1" THEN GOTO 5760 ELSE 1010
1010 ' Bild 2
1020 FOR A=0 TO 255 STEP 12
1030 LINE(A,190)-(A+10,191),13,BF
```

FRETY Teil 4
=====

```
1040 NEXT A
1050 LINE(0,186)-(255,188),12,BF
1060 FOR A=0 TO 60 STEP 12
1070 LINE(A,130)-(A+10,131),13,BF
1080 LINE(A+90,110)-(A+100,111),13,BF
1090 LINE(A+182,130)-(A+192,131),13,BF
1100 NEXT A
1110 LINE(0,126)-(70,128),12,BF
1120 LINE(90,106)-(162,108),12,BF
1130 LINE(162,110)-(162,111),13
1140 LINE(182,126)-(255,128),12,BF
1150 LINE(254,130)-(255,131),13,BF
1160 FOR A=0 TO 110 STEP 12
1170 LINE(A,70)-(A+10,71),13,BF
1180 LINE(138+A,70)-(148+A,71),13,BF
1190 NEXT A
1200 LINE(220,70)-(255,72),1,BF
1210 LINE(0,66)-(118,68),12,BF
1220 LINE(138,66)-(215,68),12,BF
1230 FOR A=1 TO 108 STEP 4
1240 LINE(74,78+A)-(86,78+A),5
1250 LINE(166,78+A)-(178,78+A),5
1260 NEXT A
1270 FOR A=1 TO 68 STEP 4
1280 LINE(122,38+A)-(134,38+A),5
1290 NEXT A
1300 LINE(123,38)-(123,105),5
1310 LINE(133,38)-(133,105),5
1320 LINE(75,78)-(75,185),5
1330 LINE(85,78)-(85,185),5
1340 LINE(167,78)-(167,185),5
1350 LINE(177,78)-(177,185),5
1360 LINE(128,112)-(128,160),14
1370 LINE(46,72)-(46,100),14
1380 LINE(210,72)-(210,100),14
1390 LINE(230,0)-(230,40),14
1400 FOR A=0 TO 44 STEP 4
```

FRETY   Teil 5
=====

```
1410 LINE(A,24)-(A,58),14
1420 NEXT A
1430 LINE(0,24)-(44,58),14,B
1440 LINE(44,58)-(60,64),14
1450 PUT SPRITE 5,(230,2),10,13
1460 ' Spielablauf
1470 DRAW"bm186,120f5e5"
1480 X=201
1490 Y=169
1500 U=0
1510 O=60
1520 P=49
1530 A$="R"
1540 Q=4
1550 I=0
1560 L=20
1570 RI=1
1580 C=13
1590 S=2
1600 S1=4
1610 A1=180
1620 I1=0
1630 GOSUB 4770
1640 INTERVAL ON
1650 ON INTERVAL=18 GOSUB 4510
1660 IF I1=0 AND Y=109 AND X<194 AND X>1
74 THEN GOSUB 6670
1670 IF I1=13 AND Y=109 AND X<66 AND X>5
0 THEN GOSUB 6670
1680 IF STICK(AZ)=3 THEN C=0:D=4:GOSUB 4
670
1690 IF STICK(AZ)=7 THEN C=13:D=-4:GOSUB
 4670
1700 IF STICK(AZ)=1 THEN Z=-2:GOSUB 4810
1710 IF STICK(AZ)=5 THEN Z=2:GOSUB 4810
1720 IF POINT(X,Y+17)=12 AND STICK(AZ)=2
 THEN C=0:RI=1:GOSUB 5000
```

FRETY Teil 6

```
1730 IF POINT(X,Y+17)=12 AND STICK(AZ)=8
 THEN C=13:RI=-1:GOSUB 5000
1740 IF T$="J" AND STRIG(AZ)=-1 THEN T$=
"N":GOSUB 5280
1750 GOTO 1660
1760 DATA 0000000000000000
1770 DATA 0000000000000000
1780 DATA 0000000000000000
1790 DATA 0000000000000000
1800 DATA 0000000110000000
1810 DATA 0000000110000000
1820 DATA 0000000110000000
1830 DATA 0000000110000000
1840 DATA 0000000110000000
1850 DATA 0000000110000000
1860 DATA 0000000110000000
1870 DATA 0000000110000000
1880 DATA 0000000110000000
1890 DATA 0000000110000000
1900 DATA 0000000110000000
1910 DATA 0000000110000000
1920 '
1930 DATA 0000001110000000
1940 DATA 0000011100000000
1950 DATA 0000010111100000
1960 DATA 0000011001100000
1970 DATA 0000011111000000
1980 DATA 0000000000000000
1990 DATA 0000011100000000
2000 DATA 0000011100000000
2010 DATA 0000011100000000
2020 DATA 0000011100000000
2030 DATA 0000011000000000
2040 DATA 0000111000000000
2050 DATA 0010000000011000
2060 DATA 0111000001110000
2070 DATA 0011000011100000
```

FRETY   Teil 7

```
2080 DATA 0001110001000000
2090 '
2100 DATA 0000001110000000
2110 DATA 0000011100000000
2120 DATA 0000010111100000
2130 DATA 0000011001100000
2140 DATA 0000011111000000
2150 DATA 0000000000000000
2160 DATA 0000011100000000
2170 DATA 0000011100000000
2180 DATA 0000011100000000
2190 DATA 0000011100000000
2200 DATA 0000011000000000
2210 DATA 0000111000000000
2220 DATA 0000000000000000
2230 DATA 0000000000000000
2240 DATA 0000111000000000
2250 DATA 0000111111000000
2260 '
2270 DATA 0000001100000000
2280 DATA 0000000000000000
2290 DATA 0000000000000000
2300 DATA 0000010000000000
2310 DATA 0000000000000000
2320 DATA 0000011110000000
2330 DATA 0000100010000000
2340 DATA 0000100011000000
2350 DATA 0000100011000000
2360 DATA 0000100011000000
2370 DATA 0000000000000000
2380 DATA 0001100111000000
2390 DATA 0011111111000000
2400 DATA 0001100111000000
2410 DATA 0000000000000000
2420 DATA 0000000000000000
2430 '
2440 DATA 0000001100000000
```

FRETY Teil 8

```
2450 DATA ØØØØØØØØØØØØØØØØ
2460 DATA ØØØØØØØØØØØØØØØØ
2470 DATA ØØØØØ1ØØØØØØØØØØ
2480 DATA ØØØØØØØØØØØØØØØØ
2490 DATA ØØØØØ1111ØØØØØØØ
2500 DATA ØØØØ1ØØØ1ØØØØØØØ
2510 DATA ØØØØ1ØØØ11ØØØØØØ
2520 DATA ØØØØ1ØØØ11ØØØØØØ
2530 DATA ØØØØ1ØØØ11ØØØØØØ
2540 DATA ØØØØØØØØØØØØØØØØ
2550 DATA ØØØØ1ØØ111ØØØØØØ
2560 DATA ØØØØ11111ØØØØØØØ
2570 DATA ØØØØ111ØØØØØØØØØ
2580 DATA ØØØØØØØØØØØØØØØØ
2590 DATA ØØØØØØØØØØØØØØØØ
2600 '
2610 DATA ØØØØØ11111ØØØØØØ
2620 DATA ØØØØ1111ØØØØØØØØ
2630 DATA ØØØØ1111111ØØØØØ
2640 DATA ØØØØ111111ØØØØØØ
2650 DATA ØØØØØ1111ØØØØØØØ
2660 DATA ØØØØØØØØØØØØØØØØ
2670 DATA ØØØØØ1111111ØØØØ
2680 DATA ØØØØØ1111111ØØØØ
2690 DATA ØØØØØ1111ØØØØØØØ
2700 DATA ØØØØØØØØØØØØØØØØ
2710 DATA ØØØØØØØØØØØØØØØØ
2720 DATA ØØØØØØØØØØØØØØØØ
2730 DATA ØØØØØØØØØØØØØØØØ
2740 DATA ØØØØØØØØØØØØØØØØ
2750 DATA ØØØØØØØØØØØØØØØØ
2760 DATA ØØØØØØØØØØØØØØØØ
2770 '
2780 DATA ØØØØØ111ØØØØØØØØ
2790 DATA ØØØØ11ØØØØØØ1ØØØ
2800 DATA ØØØØ1ØØØØØØØ1ØØØ
2810 DATA ØØØØØØ111ØØØ1ØØØ
```

FRETY   Teil 9
=====

```
2820 DATA 0000000000001000
2830 DATA 0000111110001000
2840 DATA 0001100000001000
2850 DATA 0001100000001000
2860 DATA 0001100001010000
2870 DATA 0001111111000000
2880 DATA 0001111111000000
2890 DATA 0000000000000000
2900 DATA 0010111111000000
2910 DATA 0111011111101000
2920 DATA 1111001111110000
2930 DATA 0011100011100000
2940 '
2950 DATA 0000011100000000
2960 DATA 0000110000000001
2970 DATA 0000100000000001
2980 DATA 0000001110000001
2990 DATA 0000000000000010
3000 DATA 0000111110000010
3010 DATA 0001100000000100
3020 DATA 0001100000001000
3030 DATA 0001100001010000
3040 DATA 0001111111000000
3050 DATA 0001111111000000
3060 DATA 0000000000000000
3070 DATA 0001111111000000
3080 DATA 0001111100000000
3090 DATA 0001110000000000
3100 DATA 0001111100000000
3110 '
3120 DATA 0000000000000000
3130 DATA 0000000000000000
3140 DATA 0000010001000000
3150 DATA 0000011111110000
3160 DATA 0000011111110000
3170 DATA 0000001110110000
3180 DATA 0001100000110000
```

FRETY   Teil 10
=====

```
3190 DATA 0001100000110000
3200 DATA 0001100000110000
3210 DATA 0001100000000000
3220 DATA 0001100000000000
3230 DATA 0001100000000000
3240 DATA 0000000000000000
3250 DATA 0000000000000000
3260 DATA 0000000000000000
3270 DATA 0000000000000000
3280 '
3290 DATA 0000001110000000
3300 DATA 0000011111000000
3310 DATA 0000011111000000
3320 DATA 0000000000000000
3330 DATA 0000000000000000
3340 DATA 0000011111000000
3350 DATA 0000011111000000
3360 DATA 0000111111100000
3370 DATA 0000111111100000
3380 DATA 0000111111100000
3390 DATA 0000000000000000
3400 DATA 0000111111100000
3410 DATA 0000111011110000
3420 DATA 0000111000000000
3430 DATA 0000111000000000
3440 DATA 0001111000000000
3450 '
3460 DATA 0000000000000000
3470 DATA 0000011111000000
3480 DATA 0000011111110000
3490 DATA 0000011111110000
3500 DATA 0000011111110000
3510 DATA 0000001110110000
3520 DATA 0001100000110000
3530 DATA 0001100000110000
3540 DATA 0001100000110000
3550 DATA 0001100000000000
```

FRETY   Teil 11

```
3560 DATA ØØØ11ØØØØØØØØØØØ
3570 DATA ØØØ11ØØØ111ØØØØØ
3580 DATA ØØØ11ØØØ1111ØØØØ
3590 DATA ØØØØØØØØØØØØØØØØ
3600 DATA ØØØØ111ØØØØØØØØØ
3610 DATA ØØØ1111ØØØØØØØØØ
3620 '
3630 DATA ØØØØØØØØ111ØØØØØ
3640 DATA ØØØØØØØ111ØØØØØØ
3650 DATA ØØØØØØØ1Ø1111ØØØ
3660 DATA ØØØØØØØ1ØØØ11ØØØ
3670 DATA ØØØØØØØ11111ØØØØ
3680 DATA ØØØØØØØØØØØØØ11Ø
3690 DATA ØØØØØØØ11111111Ø
3700 DATA ØØØØØØ1111111ØØØ
3710 DATA ØØØØØØØØØØØØØØØØ
3720 DATA ØØ11ØØØØØØØØØØØØ
3730 DATA ØØ11ØØØØØØØØØØØØ
3740 DATA ØØ11ØØØØØØØØØØØØ
3750 DATA ØØ1ØØØØØØØØØØØØØ
3760 DATA ØØ1ØØØØØØØØØØØØØ
3770 DATA ØØ1ØØØØØØØØØØØØØ
3780 DATA ØØØØØØØØØØØØØØØØ
3790 '
3800 DATA ØØØØØØØØ1ØØØØØØØ
3810 DATA ØØØØØØØ1ØØØØØØØØ
3820 DATA ØØØØØØØØØØØØØØØØ
3830 DATA ØØØØØØØØØØØØØØØØ
3840 DATA ØØØØØØØØØØØØØØØØ
3850 DATA ØØØØØØ11111ØØØØØ
3860 DATA ØØØØØ11ØØØØØØØØØ
3870 DATA ØØØØØ1ØØØØØØØØØØ
3880 DATA ØØØØØØ11111ØØØØØ
3890 DATA ØØØØ11Ø1111ØØØØØ
3900 DATA ØØØØ111Ø11ØØØØØØ
3910 DATA ØØØØ1111ØØØØØØØØ
3920 DATA ØØØØØØØØØØØØØØØØ
```

FRETY   Teil 12

```
3930 DATA 0000000000000000
3940 DATA 0000000000000000
3950 DATA 0000000000000000
3960 '
3970 DATA 0000011000000000
3980 DATA 0000111000000000
3990 DATA 0001010100000000
4000 DATA 0001010010000000
4010 DATA 0001001000000000
4020 DATA 0000100100000000
4030 DATA 0000000000000000
4040 DATA 0000000000000000
4050 DATA 0000000000000000
4060 DATA 0000000000000000
4070 DATA 0000000000000000
4080 DATA 0000000000000000
4090 DATA 0000000000000000
4100 DATA 0000000000000000
4110 DATA 0000000000000000
4120 DATA 0000000000000000
4130 ' links-rechts waerter
4140 GOSUB 5630
4150 O=O+Q
4160 G=G+1
4170 E=(-1)↑G
4180 IF E=-1 THEN F=6
4190 IF E=1 THEN F=7
4200 PUT SPRITE 4,(O,P),11,5+I
4210 PUT SPRITE 3,(O,P),4,F+I
4220 IF Q=4 AND POINT(O,P+17)<>12 THEN A
$="T"
4230 IF Q=-4 AND POINT(O+16,P+17)<>12 TH
EN A$="T"
4240 IF O<1 THEN O=60:P=49:I=0:Q=4
4250 IF O>238 THEN O=60:P=49:I=0
4260 IF Y=P AND O>X-8 AND O<X+8 THEN GOS
UB 6670
```

FRETY Teil 13

```
4270 RETURN
4280 ' treppe waerter
4290 GOSUB 5630
4300 IF O>130 THEN J=4 ELSE J=0
4310 P=P+2
4320 G=G+1
4330 E=(-1)↑G
4340 IF E=-1 THEN K=0
4350 IF E=1 THEN K=13
4360 PUT SPRITE 4,(O+J,P),11,8+K
4370 PUT SPRITE 3,(O+J,P),4,9+K
4380 IF POINT(8+O,P+17)=12 THEN GOSUB 44
20
4390 IF P>Y-16 AND P<Y+16 AND O+J>X-10 A
ND O+J<X+26 THEN GOSUB 6670
4400 RETURN
4410 ' richtung
4420 A$="R"
4430 IF X<128 AND P<120 THEN Q=-4:I=13
4440 IF X>128 AND P<120 THEN Q=4:I=0
4450 IF X>160 AND P>120 THEN Q=4:I=0
4460 IF X<80 AND P>120 THEN Q=-4:I=13
4470 IF X<160 AND O>150 AND P>120 THEN Q
=-4:I=13
4480 IF X>80 AND O<90 AND P>120 THEN Q=4
:I=0
4490 RETURN
4500 ' interval
4510 ZX=ZX+1
4520 IF ZX MOD 50=0 THEN LINE(46,14)-(66
,22),1,BF:DRAW "bm 46,14":COLOR 14:PRINT
 #1,USING "###";900-ZX/2
4530 IF ZX=1800 THEN GOSUB 6680
4540 CB=CB+1
4550 IF CB=9 THEN CB=1
4560 SOUND 8,16
4570 SOUND 12,6
```

FRETY   Teil 14

```
4580 SOUND 13,9
4590 SOUND 1,SD(CB)
4600 SOUND 10,16
4610 SOUND 4,2
4620 SOUND 12,5
4630 SOUND 13,8
4640 IF A$="R" THEN GOSUB 4140
4650 IF A$="T" THEN GOSUB 4290
4660 RETURN
4670 ' links-rechts-Affe
4680 X=X+D
4690 IF X>240 THEN X=X-4
4700 IF X<2 THEN X=X+4
4710 IF POINT(X-5,Y+17)<>12 AND POINT(X+
24,Y+17)<>12 THEN GOSUB 4920
4720 U=U+1
4730 V=(-1)↑U
4740 IF V=-1 THEN S=1
4750 IF V=1 THEN S=2
4760 IF STICK(AZ)<>3 AND STICK(AZ)<>7 TH
EN S=2
4770 PUT SPRITE 2,(X,Y),6,S+C
4780 PUT SPRITE 1,(X,Y),10,2+S+C
4790 RETURN
4800 ' treppe affe
4810 IF POINT(X+2,Y)=5 AND POINT(X+12,Y)
=5 THEN GOTO 4820 ELSE RETURN
4820 IF Z=2 AND POINT(X+8,Y+17)=12 THEN
RETURN
4830 Y=Y+Z
4840 U=U+1
4850 V=(-1)↑U
4860 IF V=-1 THEN S=0
4870 IF V=1 THEN S=13
4880 PUT SPRITE 2,(X,Y),10,9+S
4890 PUT SPRITE 1,(X,Y),6,10+S
4900 RETURN
```

FRETY   Teil 15
=====

```
4910 ' fall
4920 IF STICK(AZ)=7 THEN X=X-8
4930 IF STICK(AZ)=3 THEN X=X+8
4940 Y=Y+1
4950 PUT SPRITE 2,(X,Y),10,9
4960 PUT SPRITE 1,(X,Y),6,10
4970 IF POINT(X,Y+17)=12 OR POINT(X+16,Y
+17)=12 GOTO 1660
4980 GOTO 4940
4990 ' sprung
5000 FOR L=1 TO 20
5010 PUT SPRITE 1,(X+RI*L,Y-M(L)),10,12+
C
5020 PUT SPRITE 2,(X+RI*L,Y-M(L)),6,11+C
5030 IF POINT(X+RI*L+8,Y-M(L))=14 THEN G
OSUB 5100
5040 IF X+RI*L+8>O AND X+RI*L+8<O+16 AND
 Y-M(L)+16>P AND Y-M(L)+16<P+16 THEN GOT
O 6680
5050 NEXT L
5060 X=X+RI*20
5070 D=0
5080 GOTO 4670
5090 '
5100 Y=Y-8
5110 PUT SPRITE 2,(X+RI*L,Y-M(L)),10,3+C
5120 PUT SPRITE 1,(X+RI*L,Y-M(L)),6,1+C
5130 IF STICK(AZ)=5 THEN GOSUB 5190
5140 IF STICK(AZ)=1 THEN Y=Y-1
5150 IF POINT(X+RI*L+8,Y-M(L))<>14 THEN
Y=Y+1
5160 IF Y-M(L)<5 THEN BEEP:PUT SPRITE 5,
(200,200):T$="J":GOSUB 5190
5170 GOTO 5110
5180 '
5190 Y=Y+1
5200 PUT SPRITE 1,(X+RI*L,Y-M(L)),10,12+
```

FRETY   Teil 16
=====

```
C
521Ø PUT SPRITE 2,(X+RI*L,Y-M(L)),6,11+C
522Ø IF POINT(X+RI*L,Y-M(L)+17)=12 THEN
X=X+RI*8:Y=Y-M(L):GOTO 524Ø
523Ø GOTO 519Ø
524Ø PUT SPRITE 2,(X,Y),6,2+C
525Ø PUT SPRITE 1,(X,Y),1Ø,4+C
526Ø GOTO 166Ø
527Ø ' wurf
528Ø IF C=Ø THEN AS=6
529Ø IF C=13 THEN AS=-6
53ØØ PUT SPRITE 1,(X,Y),1Ø,12+C
531Ø PUT SPRITE 2,(X,Y),6,11+C
532Ø FOR FG=1 TO 2Ø
533Ø PUT SPRITE 5,(X+FG*AS+8,Y-5),1Ø,13
534Ø IF X+FG*AS+8>O AND X+FG*AS+8<O+16 A
ND Y=P THEN V$="S":GOSUB 544Ø
535Ø IF X+FG*AS+8>A1 AND X+FG*AS+8<A1+16
 AND Y=1Ø9 THEN V$="L":GOSUB 544Ø
536Ø IF X+FG*AS+8>25Ø THEN 539Ø
537Ø IF X+FG*AS+8<2 THEN 539Ø
538Ø NEXT FG
539Ø PUT SPRITE 2,(X,Y),6,S+C
54ØØ PUT SPRITE 1,(X,Y),1Ø,2+S+C
541Ø PUT SPRITE 5,(23Ø,2),1Ø,13
542Ø GOTO 175Ø
543Ø '
544Ø RT=RT+1
545Ø ER=ER+33ØØ
546Ø O=6Ø
547Ø P=49
548Ø BEEP
549Ø IF V$="S" THEN O=6Ø:P=49:Q=4:I=Ø
55ØØ IF V$="L" THEN A1=18Ø:PUT SPRITE 5,
(1ØØ,2ØØ):PUT SPRITE 6,(1ØØ,2ØØ)
551Ø COLOR 1
552Ø DRAW "bm 2Ø4,14"
```

FRETY   Teil 17
=====

```
5530 LINE(2Ø4,14)-(22Ø,22),1,BF
5540 IF FA-RT=Ø THEN ER=ER+1ØØØ:INTERVAL
 OFF:ZX=Ø:RT=Ø:ZE=ZE+.1:GOTO 576Ø
5550 LINE(5Ø,2)-(1ØØ,1Ø),1,BF
5560 COLOR 14:DRAW "bm 196,14":PRINT #1,
FA-RT
5570 LINE(5Ø,2)-(1ØØ,1Ø),1,BF
5580 DRAW "bm 52,2"
5590 PRINT #1,USING "#####";ER
5600 IF ERR>1ØØØØ THEN ER=Ø
5610 GOTO 539Ø
5620 ' waerter 2
5630 A1=A1+S1
5640 Q1=Q1+1
5650 W1=(-1)↑Q1
5660 IF W1=-1 THEN F1=6
5670 IF W1=1 THEN F1=7
5680 PUT SPRITE 7,(A1,1Ø9),11,5+I1
5690 PUT SPRITE 6,(A1,1Ø9),4,F1+I1
5700 IF A1>24Ø THEN A1=6
5710 IF A1>17Ø AND A1<18Ø THEN S1=4:I1=Ø
:DRAW "c14bm186,12Øf5e5":DRAW "c1bm56,12
Øf5e5"
5720 IF A1>6Ø AND A1<7Ø THEN S1=-4:I1=13
:DRAW"c14bm56,12Øf5e5":DRAW "c1bm186,12Ø
f5e5"
5730 IF A1<2 THEN A1=236:S1=-4:I1=13
5740 IF A1>X-8 AND Y=1Ø9 AND A1<8+X THEN
 GOSUB 667Ø
5750 RETURN
5760 ' Bild eins
5770 FOR A=1 TO 7
5780 PUT SPRITE A,(1ØØ,2ØØ)
5790 NEXT A
5800 LINE(Ø,4Ø)-(25Ø,19Ø),1,BF
5810 COLOR 1
5820 LINE(2Ø4,14)-(22Ø,22),1,BF
```

FRETY   Teil 18
=====

```
5830 COLOR 14
5840 LINE(Ø,24)-(6Ø,4Ø),1
5850 LINE(23Ø,Ø)-(23Ø,4Ø),1
5860 LINE(Ø,24)-(44,4Ø),1,BF
5870 LINE(Ø,12Ø)-(256,17Ø),1,BF
5880 LINE(Ø,3Ø)-(255,9Ø),5,BF
5890 LINE(Ø,1Ø1)-(255,12Ø),12,BF
5900 LINE(Ø,9Ø)-(255,1ØØ),1Ø,BF
5910 LINE(Ø,17Ø)-(255,191),2,BF
5920 LINE(Ø,121)-(255,121),15
5930 LINE(Ø,169)-(255,169),15
5940 B=Ø:K$="N":GOSUB 611Ø
5950 INTERVAL ON
5960 ON INTERVAL=12 GOSUB 622Ø
5970 Z=Ø
5980 IF K$="N" AND STICK(AZ)=3 THEN K$="
J":GOSUB 6Ø1Ø
5990 IF K$="J" AND STICK(AZ)=7 THEN K$="
N":GOSUB 6Ø1Ø
6000 GOTO 598Ø
6010 SOUND 9,16
6020 SOUND 13,9
6030 SOUND 12,3
6040 SOUND 3,Ø
6050 SOUND 7,B4
6060 U=U+1
6070 E=(-1)↑U
6080 IF E=-1 THEN S=Ø
6090 IF E=1 THEN S=1
6100 Z=Z+.5
6110 PUT SPRITE 2,(1ØØ+Z,13Ø),6,1+S
6120 PUT SPRITE 1,(1ØØ+Z,13Ø),1Ø,3+S
6130 A=A-4
6140 IF A=64 THEN A=Ø
6150 PUT SPRITE 5,(A+5Ø,1ØØ),15,Ø
6160 PUT SPRITE 6,(A+114,1ØØ),15,Ø
6170 PUT SPRITE 7,(A+178,1ØØ),15,Ø
```

FRETY   Teil 19
=====

```
6180 PUT SPRITE 8,(A+246,100),15,0
6190 IF 100+Z>240 THEN GOTO 6320
6200 RETURN
6210 ' Waerter
6220 RG=RG+1
6230 T=(-1)↑RG
6240 IF T=-1 THEN I=6
6250 IF T=1 THEN I=7
6260 B=B+ZE
6270 PUT SPRITE 4,(50+B,130),11,5
6280 PUT SPRITE 3,(50+B,130),4,I
6290 IF 64+B>100+Z THEN GOTO 6540
6300 RETURN
6310 ' Ende Bild 1
6320 INTERVAL OFF
6330 LINE(0,30)-(255,191),1,BF
6340 COLOR 14,1,1
6350 PUT SPRITE 1,(100,200)
6360 PUT SPRITE 2,(100,200)
6370 PUT SPRITE 3,(100,200)
6380 PUT SPRITE 4,(100,200)
6390 PUT SPRITE 5,(100,200)
6400 PUT SPRITE 6,(100,200)
6410 PUT SPRITE 7,(100,200)
6420 PUT SPRITE 8,(100,200)
6430 ER=ER+(100+Z-50+O)*10
6440 BEEP
6450 COLOR 1
6460 LINE(50,2)-(100,10),1,BF
6470 COLOR 14
6480 DRAW "bm 52,2"
6490 PRINT #1,USING "#####";ER
6500 IF ER>10000 THEN ER=0
6510 FA=FA+1
6520 F$="2"
6530 GOTO 860
6540 ' Abzug Affe 1
```

FRETY   Teil 20
=====

```
6550 Z=0
6560 B=0
6570 CV=CV-1
6580 BEEP
6590 LINE(201,2)-(220,10),1,BF
6600 COLOR 14
6610 DRAW"BM 200,2"
6620 PRINT #1,CV
6630 IF CV=0 THEN GOSUB 6800
6640 FOR R=1 TO 500
6650 NEXT R
6660 RETURN
6670 ' Abzug Affe 2
6680 CV=CV-1
6690 BEEP
6700 LINE(201,2)-(220,10),1,BF
6710 COLOR 14
6720 DRAW "bm 200,2"
6730 PRINT #1,CV
6740 IF CV=0 THEN GOSUB 6800
6750 FOR R=1 TO 500
6760 NEXT R
6770 PUT SPRITE 5,(230,2),10,13
6780 GOTO 1480
6790 ' Game over
6800 INTERVAL OFF
6810 SOUND 8,0
6820 SOUND 9,0
6830 SOUND 10,0
6840 LINE(85,80)-(168,110),4,BF
6850 COLOR 15
6860 DRAW "bm 92,90"
6870 PRINT #1,"GAME OVER"
6880 DRAW "bm 92,98"
6890 PRINT #1,"weiter->j"
6900 A$=INKEY$
6910 IF A$="j" OR A$="J" THEN RUN ELSE G
```

FRETY   Teil 21
=====

```
OTO 6900
6920 SCREEN 2
6930 DRAW "bm 12,10"
6940 PRINT #1,"Bild 1:Durch Hin- und Her
bewe-"
6950 DRAW "bm 68,18"
6960 PRINT #1,"gen des Joysticks"
6970 DRAW "bm 68,26"
6980 PRINT #1,"(bzw. der Tastatur) be-"
6990 DRAW "bm 68,34"
7000 PRINT #1,"wegen Sie den Affen"
7010 DRAW "bm 68,42"
7020 PRINT #1,"vorwaerts, um dem Waer-"
7030 DRAW "bm 68,50"
7040 PRINT #1,"ter zu entkommen."
7050 DRAW "bm 12,66"
7060 PRINT #1,"Bild 2:Das Ziel dieses Bi
ldes"
7070 DRAW "bm 68,74"
7080 PRINT #1,"ist es, Bananen zu sam-"
7090 DRAW "bm 68,82"
7100 PRINT #1,"meln und Waerter damit"
7110 DRAW "bm 68,90"
7120 PRINT #1,"zu bewerfen."
7130 DRAW "bm 68,106"
7140 PRINT #1,"Gesprungen wird mit je-"
7150 DRAW "bm 68,114"
7160 PRINT #1,"weils 2 Pfeiltasten"
7170 DRAW "bm 68,122"
7180 PRINT #1,"gleichzeitig (z.B.hoch"
7190 DRAW "bm 68,130"
7200 PRINT #1,"und rechts d.h. Sprung"
7210 DRAW "bm 68,138"
7220 PRINT #1,"nach oben rechts)."
7230 DRAW "bm 12,170"
7240 PRINT #1,"Druecken Sie die Space-Ta
ste!"
7250 IF STRIG(0)=-1 THEN GOTO 210 ELSE 7
250
```

Kurzbeschreibung zu Programm MISTER MINER

Sehr geehrter Leser!

Vor Ihnen liegt der Anfang eines langen Listings, und da die anderen nicht sehr viel kürzer sind, schätze ich, daß Sie noch keines eingetippt haben. So könnte ich noch kurz auf unsere Arbeit eingehen:

Wir haben dieses Spiel auf einem SPECTRAVIDEO-328 geschrieben, und da dies kein richtiger MSX-Computer ist, so wie wahrscheinlich Ihrer, mußte der Herr Lüers, der Autor dieses Buches, zunächst einiges umändern, um es dann zur Probe in einen MSX-Computer einzutippen. Dazu benutzte er ein Diktiergerät. Ich kann Ihnen diese gute Idee nur nahelegen, es sei denn, Sie haben jemanden, der Ihnen so diktiert, denn spätestens bei den DATAs und den DRAW-Befehlen werden Sie durcheinanderkommen. Lassen Sie den Comuter am besten bis zur letzten Zeile an. Es schadet dem Computer nicht, über einen längeren Zeitraum in Betrieb zu sein. Sollte am Ende der langen Arbeit etwas nicht stimmen, so müßten Sie nur mit dem Original im Buch vergleichen.

Zum Spiel: Ihre Spielfigur ist ein kleiner Bergarbeiter und heißt daher MR MINER. Er bewegt sich nach links und rechts, auf der Leiter natürlich auch von oben nach unten, kann springen und, hat er an der Wasserpumpe rechts unten aufgetankt, so kann er auch schießen.

Die Funktionen liegen hier ganz normal, jedoch für den Sprung muß man den Joystick nach schräg links oben bzw. schräg rechts oben legen, die Cursortasten entsprechend "rauf" und "links" bzw. "rauf" und "rechts". Der Feuerknopf bzw. die Leertaste dienen zum Schießen.

Durch das Photo haben Sie ja schon einen Eindruck. Außerdem gibt es eine Fledermaus, die ein Ei wirft, welches sich am Boden in einen Diamanten verwandelt. Für einen Diamanten gibt es 1000 Punkte, für den Diamanten links unten 2000 Punkte. Sie werden aufgesammelt, indem Sie MR MINER genau auf einen solchen Diaman-

ten dirigieren und den Hebel nach unten legen. Der nun verschwundene Diamant spiegelt sich allerdings nur für kurze Zeit in Ihrer Zeit wider. Diese Zeit startet bei 5000 und ist sie bei 0, haben Sie einen MR MINER verloren. Sie können seine Lebenszeit jedoch durch das Aufsammeln eines Diamanten verlängern, da sich 1000 Punkte zu der Zeit addieren.

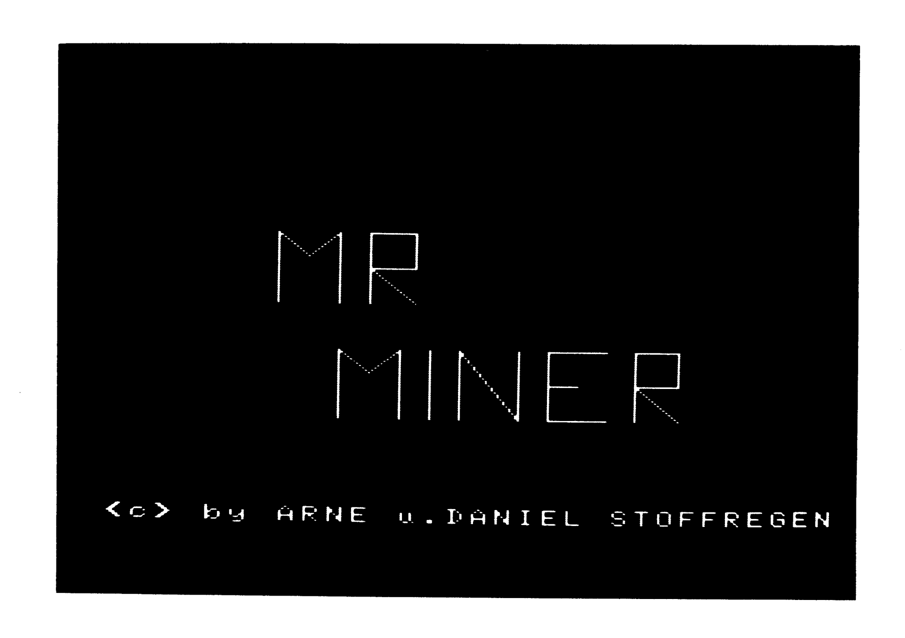
Also, Sie starten links oben und um rechts unten an der Pumpe aufzutanken, drücken Sie einfach den Feuerknopf. Dann füllt sich unten bei "WATER" ein Streifen blau. Er nimmt bei jedem Schuß natürlich ab und so bedarf es je nach Länge des Spiels mehrmaligen Auftankens.

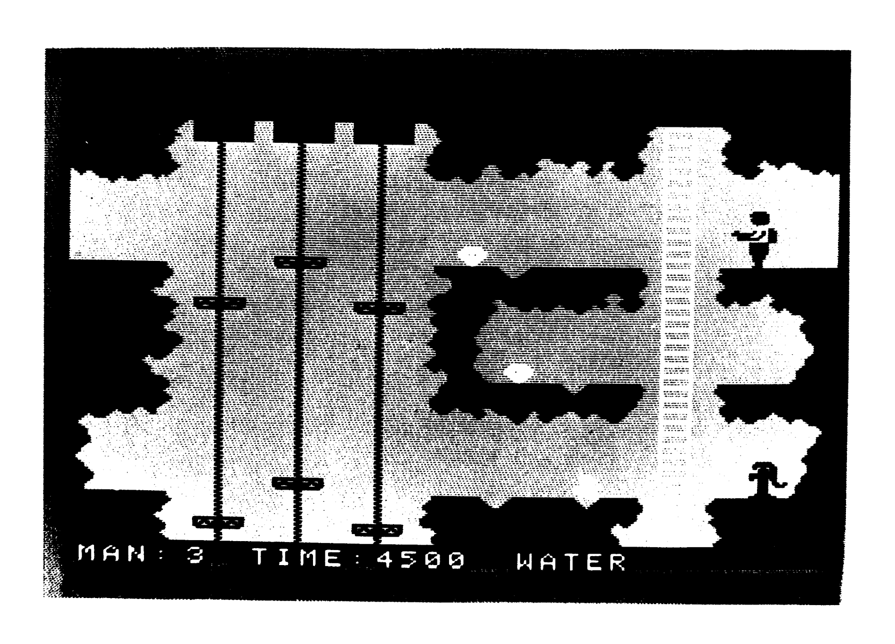
Mit dem gefüllten Tank auf dem Rücken von MR MINER kann er seine Spritzpistole bedienen. Diese ist aber nur für einen Geist gefährlich, den Sie übrigens nicht berühren dürfen. Für einen Geist gibt es 100 Punkte.

Wenn Sie jetzt glauben, Sie könnten durch den Gang rechts oben in ein zweites Bild gelangen, muß ich Sie enttäuschen. Auch der Sound, falls vorhanden, läßt zu wünschen übrig. Die Probleme beim Programmieren lagen nicht nur darin, daß zu viele Sprites auf einer Reihe waren - dann tritt nämlich eine kurzzeitige Auslöschung ein - oder der noch etwas langsamen Sprache BASIC, sondern auch in der kurzen Zeit, die wir für dieses Spiel zur Verfügung hatten. So hatten wir natürlich noch etwas geplant, doch das sollte Sie nicht davon abhalten, eventuell eine Fortsetzung, ein zweites Bild aufgrund Ihrer vielleicht in diesem Buch erworbenen Kenntnisse zu erstellen oder gar das Progrmm zu verbessern, erschweren oder abzuändern. Ich glaube, es sind Ihnen hier wenig Grenzen gesetzt und so können wir Ihnen viel Spaß und Glück wünschen.

Die Programmierer: Arne und Daniel Stoffregen

P.S.: Hätten Sie noch irgendwelche Fragen, so können Sie uns gerne schreiben:
Arne und Daniel Stoffregen, Wolteringstr.27
in Deutschland, 4400 Münster/Westfalen.


MR
MINER
<c> by ARNE u.DANIEL STOFFREGEN


MAN: 3 TIME:4500 WATER

MISTER MINER Teil 1

```
10 REM MISTER MINER
20 REM MSX Grafik und Sound
30 REM Copyright 1985 DATA BECKER &
       Rainer Lüers & Arne und
       Daniel Stoffregen
40 OPEN "grp:"FOR OUTPUT AS #1
50 GOSUB 6820
60 CLEAR 4000
70 OPEN "grp:" FOR OUTPUT AS #1
80 DEFINT A-Z
90 ON STRIG GOSUB 6360,6360,6360
100 DIM SP$(35),B(5),C(5),H$(20),H(20)
110 DATA 5,7,6,7,15,17,16,17
120 REM Laufstellungen einlesen
130 FOR T=0 TO 7
140 READ F(T)
150 NEXT T
160 VFL=5
170 GOSUB 6110
180 DRAW "a0"
190 REM Sprungoperatoren (+9,-4 usw.)
200 DATA +9,-4,+9,-2,+9,+2
210 RESTORE 200
220 FOR T=1 'TO 3
230 READ B(T),C(T)
240 NEXT T
250 SCREEN 2,2
260 GOSUB 2120 'Sprites
270 FOR T=1 TO 31
280 SPRITE$(T)=SP$(T)
290 NEXT T
300 Q$="r15d3l1d1l6d1l1u1l6u1l1u3d3r1e2f
2e2r1d1r1f1e2f2d1l6u2l1d2"
310 DATA 9,10,7,8,3,4
320 REM Sound einlesen
330 RESTORE 310
340 FOR T=1 TO 6
```

MISTER MINER   Teil 2
============

```
350 READ S(T)
360 NEXT T
370 REM Zeit definieren für Abstand
380 ON INTERVAL=100 GOSUB 6790
390 REM alle Variabln auf den
        Anfangswert setzen
400 A3=160
410 A5=96
420 A4=106
430 A6=76
440 M=4
450 O=194
460 P=-1
470 F=240
480 REM Anfangskoordinaten der Spiel-
        figur
490 X=0
500 Y=20
510 N=0
520 A1=10
530 A2=0
540 L=1
550 L1=0
560 BO=0
570 WA$="LEER"
580 LINE (190,182)-(256,192),1,BF
590 ZEIT=5000
600 MN=INT(RND(-TIME)*5)+1
610 EI$=""
620 OA=25
630 PUT SPRITE 29,(0,0),0,31
640 X10=240
650 LE$=""
660 REM Sound definieren
670 SOUND 7,&B00110100
680 SOUND 1,13
690 SOUND 12,2
```

MISTER MINER Teil 3

```
700 SOUND 11,55
710 SOUND 8,16
720 SOUND 6,245
730 SOUND 3,6
740 SOUND 9,8
750 IF LI$="N" THEN LI$="":LINE (39,9)-(
109,180),3,BF:JJK$="J":GOSUB 5410
760 PUT SPRITE 27,(0,0),0,31
770 STRIG(1) ON
780 STRIG(0) ON
790 REM Interval anstellen
800 INTERVAL ON
810 REM Diamanten in ihren verschie-
        denen Positionen erscheinen
        lassen
820 DRAW "bm15,159;"+DI$
830 P$(1)="J"
840 DRAW "bm146,109;"+DI$
850 P$(3)="J"
860 DRAW "BM130,59;"+DI$
870 P$(5)="J"
880 REM Beginn des Hauptprogramms
890 A=STICK(0)
900 IF A=0 THEN GOSUB 1560 ELSE IF FA$="
N" THEN ON A GOSUB 1670,1440,1450,1430,1
800,1430,1500,1550
910 REM Richtung (nach rechts
        oder links)
920 IF R$="1" THEN I=9 ELSE I=0
930 IF J$<>"" THEN 950 ELSE GOSUB 1990
940 REM sollen die Fahrstühle
        anhalten?
950 IF LI$="N" THEN A2=A2-4:A5=A5-4:GOTO
 1020 ELSE A1=A1+2:A2=A2-4:A3=A3+3:A4=A4
+2:A5=A5-4:A6=A6+3
960 IF A1>192-12 THEN A1=0 ELSE IF A2<0
THEN A2=192-12 ELSE IF A3>192-12 THEN A3
```

```
=0 ELSE IF A4>192-12 THEN A4=0 ELSE IF A
5<0 THEN A5=192-12 ELSE IF A6>192-12 THE
N A6=0
970 REM die Fahrstühle setzen
980 PUT SPRITE 10,(40,A1),1,24
990 PUT SPRITE 12,(92,A3),1,24
1000 PUT SPRITE 13,(40,A4),1,24
1010 PUT SPRITE 15,(92,A6),1,24
1020 IF LI$="N" THEN PUT SPRITE 27,(X10,
22),1,ER ELSE 1130
1030 REM Vogel von rechts nach links
         fliegen lassen
1040 X10=X10-5
1050 IF X10<=0 THEN X10=240:OA=25
1060 REM Flügel bewegen
1070 RE=RE+1
1080 IF RE/4<>INT(RE/4) THEN IF ER=25 TH
EN ER=26 ELSE ER=25
1090 IF EI$="J" THEN 1100 ELSE IF X10<PX
(MN) AND X10+20>PX(MN) AND P$(MN)="N" TH
EN EI$="J" ELSE MN=INT(RND(-TIME)*5)+1:G
OTO 1130
1100 PUT SPRITE 29,(PX(MN),OA),15,27
1110 OA=OA+5
1120 IF OA>=PY(MN) THEN EI$="":FG$=STR$(
PX(MN)):PUT SPRITE 29,(0,0),0,31:GF$=STR
$(PY(MN)):DRAW "bm"+FG$+","+GF$+DI$:P$(M
N)="J"
1130 PUT SPRITE 11,(66,A2),1,24
1140 PUT SPRITE 14,(66,A5),1,24
1150 IF LI$="N" THEN IF A2<0 THEN A2=180
1160 IF LI$="N" THEN IF A5<0 THEN A5=180
1170 PUT SPRITE 25,(O,P),15,M
1180 IF X>30 AND X+9<56 THEN GOSUB 2050
1190 IF X>56 AND X+9<82 THEN GOSUB 2070
1200 IF X>82 AND X+9<108 THEN GOSUB 2090
1210 REM Lauffarbe setzen bei
```

MISTER MINER   Teil 5
============

```
        Leitersprung
1220 IF X>185 AND X+16<222 THEN V=11 ELS
E V=12
1230 REM Geist bewegen
1240 IF H$(L)="L" THEN O=O-10 ELSE IF H$
(L)="R" THEN O=O+10 ELSE IF H$(L)="O" TH
EN P=P-10 ELSE IF H$(L)="U" THEN P=P+10
1250 L1=L1+1
1260 IF L1>H(L) THEN L1=1:L=L+1:IF H$(L)
="E" THEN O=194:P=0:L=1:L1=0
1270 IF M=4 THEN M=14 ELSE M=4
1280 REM Sprung nach rechts
1290 IF J$="R" THEN X=X+B(J1):Y=Y+C(J1):
J1=J1+1:IF J1=3 THEN J$="":J1=0
1300 IF SCHLUSS=1 THEN GOSUB 6380
1310 REM Sprung nach links
1320 IF J$="L" THEN X=X-B(J2):Y=Y+C(J2):
J2=J2+1:IF J2=3 THEN J$="":J2=0
1330 IF O>=X-4 AND O<=X+6 AND P>=Y-4 AND
 P<=Y+17 THEN 6270
1340 REM PUTSPRITE für Männchensprung
1350 GOSUB 1580
1360 REM Fahrstühle halten an ja/nein
1370 BQ=BQ+1
1380 IF BQ=209 THEN GOSUB 6520
1390 DF=DF+1
1400 IF DF=7 THEN DF=1
1410 SOUND 3,S(DF)
1420 GOTO 890
1430 RETURN
1440 IF LE$="J" OR JP$="J" THEN RETURN E
LSE J$="R":R$="r":JP$="J":RETURN
1450 IF LE$="J" THEN LE$="":U=7:R$="r":R
ETURN ELSE SOUND 13,9:R=R+1:IF R=4 THEN
R=0
1460 U=F(R)
1470 X=X+4
```

MISTER MINER   Teil 6
============

```
1480 R$="r"
1490 RETURN
1500 IF LE$="J" THEN LE$="":U=16:R$="l":
RETURN ELSE SOUND 13,9:K=K+1:IF K=4 THEN
 K=0
1510 U=F(K+4)
1520 X=X-4
1530 R$="l"
1540 RETURN
1550 IF LE$="J" OR JP$="J" THEN RETURN E
LSE J$="L":R$="l":JP$="J":RETURN
1560 IF R$="l" THEN U=17 ELSE U=7:RETURN
1570 RETURN
1580 IF R$="l" THEN I=10 ELSE I=0
1590 REM Richtung beim Sprung
         bestimmen
1600 IF J$="R" THEN U=8 ELSE IF J$="L" T
HEN U=18
1610 IF LE$="J" THEN RETURN
1620 PUT SPRITE 0,(X,Y),1,1+I
1630 PUT SPRITE 1,(X,Y),8,3+I
1640 PUT SPRITE 2,(X,Y+16),4,U
1650 PUT SPRITE 3,(X,Y),7,2+I:RETURN
1660 REM Leiter hochgehen
1670 IF X<191 OR X>205 THEN RETURN
1680 REM Darstellung von hinten
1690 Y=Y-5
1700 X=192
1710 PUT SPRITE 0,(X,Y),1,21
1720 PUT SPRITE 1,(X,Y),8,22
1730 PUT SPRITE 2,(X,Y),7,23
1740 REM Beine bewegen
1750 G=G+1
1760 IF G/2<>INT(G/2) THEN G1=9 ELSE G1=
19:G=0
1770 PUT SPRITE 3,(X,Y+16),4,G1
1780 LE$="J"
```

MISTER MINER   Teil 7

```
1790 RETURN
1800 REM Leiter runter oder Diamant
         aufsammeln
1810 IF X>191 AND X<205 THEN 1890
1820 IF X>5 AND X<12 AND Y=133 AND P$(1)
="J" THEN LINE(11,140)-(20,159),3,BF:P$(
1)="N":ZEI=ZEI+2000:RETURN
1830 IF X>115 AND X<123 AND Y=133 AND P$
(2)="J" THEN LINE(121,140)-(130,159),3,B
F:P$(2)="N":ZEI=ZEI+1000:RETURN
1840 IF X>136 AND X<143 AND Y=83 AND P$(
3)="J" THEN LINE(142,90)-(151,109),3,BF:
P$(3)="N":ZEI=ZEI+1000:RETURN
1850 IF X>220 AND X<227 AND Y=83 AND P$(
4)="J" THEN LINE(226,90)-(235,109),3,BF:
P$(4)="N":ZEI=ZEI+1000:RETURN
1860 IF X>120 AND X<127 AND Y=33 AND P$(
5)="J" THEN LINE(126,40)-(135,59),3,BF:P
$(5)="N":ZEI=ZEI+1000:RETURN
1870 RETURN
1880 REM Darstellung von hinten
1890 Y=Y+5
1900 X=192
1910 PUT SPRITE 0,(X,Y),1,21
1920 PUT SPRITE 1,(X,Y),8,22
1930 PUT SPRITE 2,(X,Y),7,23
1940 G=G+1
1950 IF G/2<>INT(G/2) THEN G1=9 ELSE G1=
19:G=0
1960 PUT SPRITE 3,(X,Y+16),4,G1
1970 LE$="J"
1980 RETURN
1990 IF POINT(X+8,Y+27)=V THEN FA$="N":N
=0:JP$="":RETURN ELSE FA$="J"
2000 FOR T=1 TO 5
2010 Y=Y+1
2020 IF POINT(X+8,Y+27)<>V THEN NEXT T E
```

MISTER MINER   Teil 8
============

```
LSE T=6:FA$="N"
2Ø3Ø IF N=11 THEN 627Ø ELSE N=N+1
2Ø4Ø RETURN
2Ø5Ø IF Y+26<A1 OR Y+26>A1+5 THEN 2Ø6Ø E
LSE Y=A1-26:FA$="N":N=Ø:JP$="":RETURN
2Ø6Ø IF Y+26<A4 OR Y+26>A4+5 THEN RETURN
 ELSE Y=A4-26:FA$="N":N=Ø:JP$="":RETURN
2Ø7Ø IF Y+26<A2 OR Y+26>A2+9 THEN 2Ø8Ø E
LSE Y=A2-26:FA$="N":N=Ø:JP$="":RETURN
2Ø8Ø IF Y+26<A5 OR Y+26>A5+9 THEN RETURN
 ELSE Y=A5-26:FA$="N":N=Ø:JP$="":RETURN
2Ø9Ø IF Y+26<A3 OR Y+26>A3+7 THEN 21ØØ E
LSE Y=A3-26:FA$="N":N=Ø:RETURN
21ØØ IF Y+26<A6 OR Y+26>A6+7 THEN RETURN
 ELSE Y=A6-26:FA$="N":N=Ø:JP$="":RETURN
211Ø GOTO 211Ø
212Ø S$=SPACE$(32)
213Ø RESTORE 252Ø
214Ø FOR T=1 TO 1Ø
215Ø FOR N=1 TO 16
216Ø READ A$
217Ø MID$(S$,N,1)=CHR$(VAL("&b"+LEFT$(A$
,8)))
218Ø MID$(S$,N+16)=CHR$(VAL("&b"+RIGHT$(
A$,8)))
219Ø NEXT N
22ØØ SP$(T)=S$
221Ø S$=SPACE$(32)
222Ø NEXT T
223Ø RESTORE 252Ø
224Ø GOSUB 541Ø
225Ø S$=SPACE$(32)
226Ø FOR T=11 TO 2Ø
227Ø FOR N=1 TO 16
228Ø READ A$
229Ø B$=SPACE$(16)
23ØØ FOR U=1 TO LEN(A$)
```

MISTER MINER   Teil 9

```
231Ø MID$(B$,LEN(A$)+1-U,1)=MID$(A$,U,1)
232Ø NEXT U:A$=B$
233Ø MID$(S$,N,1)=CHR$(VAL("&b"+LEFT$(A$
,8)))
234Ø MID$(S$,N+16)=CHR$(VAL("&b"+RIGHT$(
A$,8)))
235Ø NEXT N
236Ø SP$(T)=S$
237Ø S$=SPACE$(32)
238Ø NEXT T
239Ø RESTORE 422Ø
24ØØ S$=SPACE$(32)
241Ø FOR T=21 TO 27
242Ø FOR N=1 TO 16
243Ø READ A$
244Ø MID$(S$,N,1)=CHR$(VAL("&b"+LEFT$(A$
,8)))
245Ø MID$(S$,N+16)=CHR$(VAL("&b"+RIGHT$(
A$,8)))
246Ø NEXT N
247Ø SP$(T)=S$
248Ø S$=SPACE$(32)
249Ø NEXT T
25ØØ RETURN
251Ø 'Sprite 1
252Ø DATA ØØØØØØØØØØØØØØØØ
253Ø DATA ØØØØØØØØØØØØØØØØ
254Ø DATA ØØØØØ1111ØØØØØØØ
255Ø DATA ØØØØ111ØØ1ØØØØØØ
256Ø DATA ØØØØ111Ø1ØØØØØØØ
257Ø DATA ØØØØ11ØØØØØØØØØØ
258Ø DATA ØØØØ1ØØ11ØØØØØØØ
259Ø DATA ØØØØØØØØØØØØØØØØ
26ØØ DATA ØØØØØØØØØØØØØØØØ
261Ø DATA ØØØØØØØØØØØØØØØØ
262Ø DATA ØØØØØ1ØØ1ØØØØØ1Ø
263Ø DATA ØØØØØ1ØØØ1111111
```

MISTER MINER   Teil 10

```
2640 DATA 0000001000010100
2650 DATA 0000000111001100
2660 DATA 0000000000000000
2670 DATA 0000111111000000
2680 ' Sprite 2
2690 DATA 0000000000000000
2700 DATA 0000000000000000
2710 DATA 0000000000000000
2720 DATA 0000000000000000
2730 DATA 0000000000000000
2740 DATA 0000000000000000
2750 DATA 0000000000000000
2760 DATA 0000000000000000
2770 DATA 0000111000000000
2780 DATA 0000111100000000
2790 DATA 0000101101000000
2800 DATA 0000101111000000
2810 DATA 0000110111000000
2820 DATA 0000111000000000
2830 DATA 0000111111000000
2840 DATA 0000000000000000
2850 ' Sprite 3
2860 DATA 0000000000000000
2870 DATA 0000000000000000
2880 DATA 0000000000000000
2890 DATA 0000000110000000
2900 DATA 0000001100000000
2910 DATA 0000001111000000
2920 DATA 0000011000000000
2930 DATA 0000011100000000
2940 DATA 0000000000000000
2950 DATA 0011000000000000
2960 DATA 0111000000000000
2970 DATA 0111000000000000
2980 DATA 0111000000101000
2990 DATA 0111000000110000
3000 DATA 0111000000000000
```

MISTER MINER   Teil 11
============

```
3010 DATA 0000000000000000
3020 ' Sprite 4
3030 DATA 0000111000000000
3040 DATA 0000011100000000
3050 DATA 0000101010000000
3060 DATA 0000111110000000
3070 DATA 0000011100100000
3080 DATA 0011111111100000
3090 DATA 0010111110000000
3100 DATA 0000111000000000
3110 DATA 0000011100000000
3120 DATA 0000001110000000
3130 DATA 0000000000000000
3140 DATA 0000000000000000
3150 DATA 0000000000000000
3160 DATA 0000000000000000
3170 DATA 0000000000000000
3180 DATA 0000000000000000
3190 ' Sprite 5
3200 DATA 0000111111100000
3210 DATA 0000111111100000
3220 DATA 0000111111110000
3230 DATA 0000011111111000
3240 DATA 0000011101111100
3250 DATA 0000001100011100
3260 DATA 0000011100111000
3270 DATA 0000011001110000
3280 DATA 0000011000100000
3290 DATA 0000011100010000
3300 DATA 0000000000000000
3310 DATA 0000000000000000
3320 DATA 0000000000000000
3330 DATA 0000000000000000
3340 DATA 0000000000000000
3350 DATA 0000000000000000
3360 ' Sprite 6
3370 DATA 0000111111000000
```

MISTER MINER   Teil 12
============

```
3380 DATA ØØØØ1111111ØØØØØ
3390 DATA ØØØØ1111111ØØØØØ
34ØØ DATA ØØØØØ111111ØØØØØ
341Ø DATA ØØØØØ111111ØØØØØ
342Ø DATA Ø1111111Ø11ØØØØØ
343Ø DATA Ø11111ØØØ11ØØØØØ
344Ø DATA Ø1ØØØØØØ11ØØØØØØ
345Ø DATA ØØØØØØØØ11ØØØØØØ
346Ø DATA ØØØØØØØØ111ØØØØØ
347Ø DATA ØØØØØØØØØØØØØØØØ
348Ø DATA ØØØØØØØØØØØØØØØØ
349Ø DATA ØØØØØØØØØØØØØØØØ
35ØØ DATA ØØØØØØØØØØØØØØØØ
351Ø DATA ØØØØØØØØØØØØØØØØ
352Ø DATA ØØØØØØØØØØØØØØØØ
353Ø ' Sprite 7
354Ø DATA ØØØØ111111ØØØØØØ
355Ø DATA ØØØØ111111ØØØØØØ
356Ø DATA ØØØØ111111ØØØØØØ
357Ø DATA ØØØØ111111ØØØØØØ
358Ø DATA ØØØØØ1111ØØØØØØØ
359Ø DATA ØØØØØ1111ØØØØØØØ
36ØØ DATA ØØØØØØ111ØØØØØØØ
361Ø DATA ØØØØØØ11ØØØØØØØØ
362Ø DATA ØØØØØØ11ØØØØØØØØ
363Ø DATA ØØØØØØ111ØØØØØØØ
364Ø DATA ØØØØØØØØØØØØØØØØ
365Ø DATA ØØØØØØØØØØØØØØØØ
366Ø DATA ØØØØØØØØØØØØØØØØ
367Ø DATA ØØØØØØØØØØØØØØØØ
368Ø DATA ØØØØØØØØØØØØØØØØ
369Ø DATA ØØØØØØØØØØØØØØØØ
37ØØ ' Sprite 8
371Ø DATA ØØØØ111111ØØØØØØ
372Ø DATA ØØØ11111111ØØØØØ
373Ø DATA ØØØ111111111ØØØØ
374Ø DATA ØØØØ111Ø1111ØØØØ
```

MISTER MINER   Teil 13

```
3750 DATA 0000000111100000
3760 DATA 0000011111000000
3770 DATA 0000011000000000
3780 DATA 0000010000000000
3790 DATA 0000000000000000
3800 DATA 0000000000000000
3810 DATA 0000000000000000
3820 DATA 0000000000000000
3830 DATA 0000000000000000
3840 DATA 0000000000000000
3850 DATA 0000000000000000
3860 DATA 0000000000000000
3870 ' Sprite 9
3880 DATA 0000111111110000
3890 DATA 0000111111110000
3900 DATA 0000011111100000
3910 DATA 0000011111100000
3920 DATA 0000011101100000
3930 DATA 0000011000000000
3940 DATA 0000110000000000
3950 DATA 0000011000000000
3960 DATA 0000000000000000
3970 DATA 0000000000000000
3980 DATA 0000000000000000
3990 DATA 0000000000000000
4000 DATA 0000000000000000
4010 DATA 0000000000000000
4020 DATA 0000000000000000
4030 DATA 0000000000000000
4040 ' Sprite 25
4050 DATA 1111111100000010
4060 DATA 0001111111111001
4070 DATA 0000000000000100
4080 DATA 0000000000010110
4090 DATA 0000000000000110
4100 DATA 0000000000000000
4110 DATA 0000000000000001
```

MISTER MINER   Teil 14
============

```
4120 DATA 0000000000000000
4130 DATA 0000000000000000
4140 DATA 0000000000000000
4150 DATA 0000000000000000
4160 DATA 0000000000000000
4170 DATA 0000000000000000
4180 DATA 0000000000000000
4190 DATA 0000000000000000
4200 DATA 0000000000000000
4210 ' Sprite 10 (schwarz)
4220 DATA 0111111111011110
4230 DATA 0100111111110010
4240 DATA 0000001111000000
4250 DATA 0000011111100000
4260 DATA 0000001111000000
4270 DATA 0000001111000000
4280 DATA 0000001111000000
4290 DATA 0000000000000000
4300 DATA 0000000000000000
4310 DATA 0000000000000000
4320 DATA 0000000000000000
4330 DATA 0000000000000000
4340 DATA 0000000000000000
4350 DATA 0000000000000000
4360 DATA 0000000000000000
4370 DATA 0000000000000000
4380 ' Sprite 11 (rot)
4390 DATA 0000000000000000
4400 DATA 0011000000001100
4410 DATA 0011000000001100
4420 DATA 0000000000000000
4430 DATA 0000010000100000
4440 DATA 0000010000100000
4450 DATA 0000000000000000
4460 DATA 0000000000000000
4470 DATA 0000001111000000
4480 DATA 0000001111000000
```

MISTER MINER   Teil 15
============

```
4490 DATA 0000011111100000
4500 DATA 0000011111100000
4510 DATA 0000010110100000
4520 DATA 0000010110100000
4530 DATA 0000011111100000
4540 DATA 0000000000000000
4550 ' Sprite 12 (hellblau)
4560 DATA 0000000000000000
4570 DATA 0000000000000000
4580 DATA 0000000000000000
4590 DATA 0011000000001100
4600 DATA 0011000000001100
4610 DATA 0011000000001100
4620 DATA 0011001111001100
4630 DATA 0011101111011100
4640 DATA 0001110000111000
4650 DATA 0001110000111000
4660 DATA 0000100000010000
4670 DATA 0000100000010000
4680 DATA 0000101001010000
4690 DATA 0000001001000000
4700 DATA 0000000000000000
4710 DATA 0000000000000000
4720 ' Sprite 22
4730 DATA 1111111111111111
4740 DATA 1001000110001001
4750 DATA 1010101111010101
4760 DATA 1100010110100011
4770 DATA 0111111111111110
4780 DATA 0000000110000000
4790 DATA 0000000000000000
4800 DATA 0000000000000000
4810 DATA 0000000000000000
4820 DATA 0000000000000000
4830 DATA 0000000000000000
4840 DATA 0000000000000000
4850 DATA 0000000000000000
```

MISTER MINER Teil 16

```
4860 DATA ØØØØØØØØØØØØØØØØ
4870 DATA ØØØØØØØØØØØØØØØØ
4880 DATA ØØØØØØØØØØØØØØØØ
4890 ' Sprite 23
4900 DATA ØØØØØØØØØØØØØØØØ
4910 DATA ØØØØØØØØØØØØØØØØ
4920 DATA ØØØØØØØØØØØØØØØØ
4930 DATA ØØØØØØØØØØØØØØØØ
4940 DATA ØØØØØØØØØØØØØØØØ
4950 DATA ØØØØØØØØØ1111111
4960 DATA ØØØØ11ØØ1111111Ø
4970 DATA ØØØ111Ø1111111ØØ
4980 DATA ØØ1Ø111111111ØØØ
4990 DATA Ø1111111111111ØØ
5000 DATA ØØØ111Ø111111Ø11
5010 DATA ØØØØØØØØØØØØØØØØ
5020 DATA ØØØØØØØØØØØØØØØØ
5030 DATA ØØØØØØØØØØØØØØØØ
5040 DATA ØØØØØØØØØØØØØØØØ
5050 DATA ØØØØØØØØØØØØØØØØ
5060 ' Sprite 24
5070 DATA ØØØØØØØØØØØØØØØØ
5080 DATA ØØØØØØØØØØØØØØØØ
5090 DATA ØØØØØØØØØØØØØØØØ
5100 DATA ØØØØØØØØØØØØØØØØ
5110 DATA ØØØØØØØØØØØØØØØØ
5120 DATA ØØØØØØØØØØØØØØØØ
5130 DATA ØØØØ11ØØØØØØØØØØ
5140 DATA ØØØ111Ø111111ØØØ
5150 DATA ØØ1Ø1111111111ØØ
5160 DATA Ø11111111111111Ø
5170 DATA ØØ1111Ø111111111
5180 DATA ØØØØØØØØØØØ11111
5190 DATA ØØØØØØØØØØØØØØØØ
5200 DATA ØØØØØØØØØØØØØØØØ
5210 DATA ØØØØØØØØØØØØØØØØ
5220 DATA ØØØØØØØØØØØØØØØØ
```

MISTER MINER   Teil 17
============

```
5230 ' Sprite 25
5240 DATA 0001000000000000
5250 DATA 0010100000000000
5260 DATA 0111010000000000
5270 DATA 1111101000000000
5280 DATA 1111011000000000
5290 DATA 0111110000000000
5300 DATA 0011100000000000
5310 DATA 0000000000000000
5320 DATA 0000000000000000
5330 DATA 0000000000000000
5340 DATA 0000000000000000
5350 DATA 0000000000000000
5360 DATA 0000000000000000
5370 DATA 0000000000000000
5380 DATA 0000000000000000
5390 DATA 0000000000000000
5400 REM Hintergrundgrafik
5410 IF JJK$="J" THEN 5430 ELSE COLOR 1,
3,2:CLS
5420 REM Fahrstuhltaue setzen
5430 FOR T=10 TO 182 STEP 2
5440 PSET(48,T),1
5450 NEXT T
5460 FOR T=9 TO 181 STEP 2
5470 PSET(47,T),1
5480 NEXT T
5490 FOR T=10 TO 182 STEP 2
5500 PSET(74,T),1
5510 NEXT T
5520 FOR T=9 TO 181 STEP 2
5530 PSET(73,T),1
5540 NEXT T
5550 FOR T=10 TO 182 STEP 2
5560 PSET(100,T),1
5570 NEXT T
5580 FOR T=9 TO 181 STEP 2
```

MISTER MINER   Teil 18

```
5590 PSET(99,T),1
5600 NEXT T
5610 IF JJK$="J" THEN JJK$="":RETURN
5620 LINE(74-9,184)-(74+9,192),1,BF
5630 LINE(74-9,0)-(74+9,8),1,BF
5640 LINE(48-9,184)-(48+9,192),1,BF
5650 LINE(48-9,0)-(48+9,8),1,BF
5660 LINE(100-9,184)-(100+9,192),1,BF
5670 LINE(100-9,0)-(100+9,8),1,BF
5680 FOR T=1 TO 192 STEP 5
5690 LINE(195,T)-(205,T+5),11,B
5700 NEXT T
5710 DRAW"bm0,60;c12r30"
5720 DRAW"bm5,160;c12r25"
5730 DRAW"bm118,160;c12r15f5r2e5r20f5e5r
12"
5740 DRAW"bm118,60;c12r20f5r2e5r37"
5750 DRAW"bm135,110;c12r25f5e5r17"
5760 DRAW"bm215,60;c12r50"
5770 DRAW"bm215,110;c12r25"
5780 DRAW"bm215,160;c12r25"
5790 DRAW"bm0,20;c2f3e2r1f3r2f2e3f2r2e3r
2f2e2f1e4h3u2h2e3u2erh2e4"
5800 PAINT(10,10),2
5810 DRAW"bm0,61;c2r30f2d2g3d212g2f3r3f3
g3f2g4d1f2r2f2d2g3d1g312h2g3f3d2f2r2f2d2
g2f3g5d211h311g312h2g413h2g311d2f3d2f1d2
g3d1g2d1f3d1f3d2g3d4r26f2g3d211g2f4d2g2f
3d2g2f3d1f1d1"
5820 PAINT(10,100),2
5830 DRAW"bm115,0;c2f3d2f2g3d2g2d2f1g2f4
r4e2r1f3r1e3u2e3r2f3g1f2r2e3r1f2e3h2e3r2
f2r3f2e3r2f3g2f2e3u2e2r3d2g2f4r2e3r2u2e1
h3u2e3u1e2h2e3"
5840 PAINT(150,5),2
5850 DRAW"bm215,0;c2f3d2g2d1g1d2f2g2f4r2
f2r2e3h2e3f2r1f2e3r2f3r2f2r2e2f3e2f1"
```

MISTER MINER   Teil 19

```
5860 PAINT(250,5),2
5870 DRAW"bm212,192;e3u2e2u1h2u2e3h2u2h3
e2h3u2e2r27u2e2u3e2u2e2u3h2u1e2u2e3u2e2u
1h3l2h3l2g2h3l1h2g3l2g2l2h3g3hu1h2u2e2u1
h4u1e1r25u2e2u3e3u2e3u3h3u2h2u2e2h3l2h3g
2h3e2h3g3l2g2l2h2g3h3u2e3u1h3e2h1r50"
5880 PAINT(250,185),2
5890 DRAW"bm115,192;c2e3u2e3u1h3u2e2h3l2
h1u2e3h2u1e3r15f5r2e5r20f5e5r12f2d2g2d2g
3l1h2g3d2f3d2f3r2d2f3d2f1"
5900 PAINT(160,170),2
5910 DRAW"bm187,61;c2l37g5l2h5l20df2r2f2
d3g3l2g3d2f3g2d2f3g2d2g2f3d2g4f3e2f3d3g3
d2g3d3f3r2e3r1f2r2f3e2h2e1r3f3r2f4e3u1e2
r1f2g1f2r1e3r2e2r2f3r1e2r2f2r1f2r2e3u1e3
u2e2u2e1h1l18g5h5l25u2h2u1h3u2e1u2e2u1h2
u2h1e2u2e3u2e3u1h2e2r2f3r2f2r2e2r1f2r2e3
r1f2r2f2r1e2r1
5920 DRAW"bm187,61;c2f2d4g3d2f2d1g3l2h3e
2h3l2g3f2g2h2l1g2"
5930 PAINT(126,63),2
5940 DRAW"bm227,159;c1r8l2u1l4r1u8l3d1u2
r3u2r1u2l3d1u1r1u1r4f2d7r1d1r2e1g1l2u1l1
u2l1u6l3d1l3r2l1u1l"
5950 LINE(0,182)-(256,192),1,BF
5960 DRAW "bm 3,185
5970 ZEIT=5000
5980 COLOR 15
5990 PRINT #1,"MAN:    TIME:";
6000 PRINT #1,USING "####";ZEIT;
6010 PRINT #1,"  WATER"
6020 COLOR 1
6030 DI$="c15h4u1e2r4f2d1g3h1u1l1h1l1r1u
1e1f1e1f1d1r1l1g1l1"
6040 VFL=6
6050 AR$="GH"
6060 COLOR 15
```

MISTER MINER   Teil 20
============

```
6070 GOSUB 6270
6080 COLOR 1
6090 RETURN
6100 REM Geistflugbahn
6110 DATA "U",9,"L",6,"R",6,"U",5,"L",19
,"R",3,"O",10,"R",16,"U",14,"E",0
6120 RESTORE 6110
6130 FOR T=1 TO 10
6140 READ H$(T),H(T)
6150 NEXT T
6160 REM Diamantenpositionen
6170 DATA 15,159,125,159,146,109,230,109
,130,59
6180 RESTORE 6170
6190 FOR T=1 TO 5
6200 READ PX(T),PY(T)
6210 NEXT T
6220 FOR T=1 TO 5
6230 P$(T)="N"
6240 NEXT T
6250 RETURN
6260 REM wenn man ein Männchen
         verloren hat
6270 VFL=VFL-1
6280 IF VFL=0 THEN LINE(35,185)-(46,192)
,1,BF:DRAW "bm 30,185":PRINT #1,VFL:LINE
 (90,94)-(90+16*6,106),3,BF:DRAW "bm 90,
96":PRINT #1,"  GAME OVER  ":INTERVAL OF
F:GOTO 7010
6290 LINE(35,185)-(46,192),1,BF
6300 DRAW "bm 30,185"
6310 PRINT #1,VFL
6320 IF AR$="GH" THEN AR$="":RETURN
6330 GOTO 400
6340 RETURN
6350 REM Auftanken
6360 IF X>220 AND X<230 AND Y>130 AND Y<
```

MISTER MINER   Teil 21
============

```
140 THEN GOSUB 6700:RETURN ELSE SCHUSS=1
:RETURN
6370 REM schießen bzw. spritzen
6380 IF LE$="J" THEN RETURN
6390 IF WA$="LEER" THEN RETURN
6400 IF R$="r" THEN PUT SPRITE 20,(X+16,
Y+10),4,10:GOSUB 6450 ELSE PUT SPRITE 20
,(X-16,Y+10),4,20:GOSUB 6470
6410 SCHUSS=0
6420 GOSUB 6660
6430 PUT SPRITE 20,(0,0),0,31
6440 RETURN
6450 IF O>X+14 AND O<X+33 AND P>Y+8 AND
P<Y+19 THEN L=1:L1=0:O=194:P=-1:ZEIT=ZEI
T+100:RETURN
6460 RETURN
6470 IF O>X-17 AND O<X+2 AND P>Y+8 AND P
<Y+19 THEN L=1:L1=0:O=194:P=-1:ZEIT=ZEIT
+100:RETURN
6480 RETURN
6490 SCHUSS=0
6500 RETURN
6510 REM Fahrstühle anhalten
6520 LI$="N"
6530 TR$=STR$(A1)
6540 DRAW "bm40,"+TR$+";c1"+Q$
6550 PUT SPRITE 10,(0,0),15,31
6560 TR$=STR$(A3)
6570 DRAW "bm92,"+TR$+";c1"+Q$
6580 PUT SPRITE 12,(0,0),15,31
6590 TR$=STR$(A4)
6600 DRAW "bm40,"+TR$+";c1"+Q$
6610 PUT SPRITE 13,(0,0),15,31
6620 TR$=STR$(A6)
6630 DRAW "bm92,"+TR$+";c1"+Q$
6640 PUT SPRITE 15,(255,191),15,31
6650 RETURN
```

MISTER MINER   Teil 22
============

```
666Ø IF WA=51 THEN WA$="LEER":RETURN
667Ø LINE(248-WA,186)-(248,188),1,BF
668Ø WA=WA+1
669Ø RETURN
67ØØ IF WA$<>"LEER" THEN RETURN
671Ø WA$=""
672Ø FOR T=1 TO 5Ø
673Ø PSET(248-T,186),2
674Ø PSET(248-T,187),7
675Ø PSET(248-T,188),2
676Ø NEXT T
677Ø WA=Ø
678Ø RETURN
679Ø ZEIT=ZEIT-1ØØ
68ØØ IF ZEIT<=Ø THEN 627Ø ELSE LINE(96,1
85)-(128,192),1,BF:DRAW "bm 1ØØ,185":COL
OR 15:PRINT #1,USING "####";ZEIT:RETURN
681Ø REM Anfangsgrafik
682Ø SCREEN 2
683Ø COLOR 15,1,1
684Ø CLS
685Ø DRAW "bm6Ø,4Ø;c15d3Øu3Øf1Øe1Ød3Ø"
686Ø DRAW "bm9Ø,7Ø;c15u3Ør15d15l15"
687Ø LINE(9Ø,55)-(1Ø5,7Ø),15
688Ø DRAW"bm8Ø,9Ø;d3Øu3Øf1Øe1Ød3Ø"
689Ø DRAW"bm11Ø,9Ø;d3Ø"
69ØØ DRAW"bm12Ø,9Ø;d3Ø"
691Ø DRAW"bm14Ø,9Ø;d3Ø"
692Ø LINE(12Ø,9Ø)-(14Ø,12Ø),15
693Ø DRAW"bm15Ø,9Ø;r2Øl2Ød15r1Øl1Ød15r2Ø
"
694Ø DRAW"bm18Ø,12Ø;u3Ør15d15l15"
695Ø LINE(18Ø,1Ø5)-(195,12Ø),15
696Ø DRAW "bm 5,16Ø"
697Ø PRINT #1,"<c> by ARNE u.DANIEL STOF
FREGEN"
698Ø FOR T=1 TO 1ØØØ
```

MISTER MINER Teil 23

```
6990 NEXT T
7000 RETURN
7010 FOR T=1 TO 2000
7020 NEXT T
7030 COLOR 15,1,1
7040 SCREEN 0
7050 LOCATE 10,12,0
7060 PRINT "ANOTHER GAME  ?"
7070 LOCATE 10,15
7080 PRINT "<press ENTER>"
7090 INPUT A$
7100 RUN
```

Kurzbeschreibung zu Maschinensprachroutine 1 bis 3

Ihr MSX-Computer bietet Ihnen neben dem beachtlichen BASIC-Wortschatz auch die Möglichkeit, ihn in Maschinensprache zu programmieren. Eigentlich braucht man zum Programmieren in Maschinensprache ein Pendant zum BASIC, den ASSEMBLER.

Da wir in diesem Buch keinen ASSEMBLER zum Selbsteintippen abgedruckt haben, müssen wir die Maschinensprachdaten in Form von Hexadezimalziffern eintippen. Die Bedeutung der einzugebenden Ziffern haben wir allerdings in den folgenden drei Listings ausführlich erklärt.

Worum geht es in den Listings?

Maschinensprachroutine 1 vermittelt Ihnen einen ersten Eindruck von der Schnelligkeit, die mit Maschinensprache möglich ist: Der gesamte Textbildschirm wird in Bruchteilen von einer Sekunde mit dem Buchstaben 'M' gefüllt. Sie können dieses Maschinensprachprogramm auch so abändern, daß Ihr MSX-Computer einen Bereich auf dem Grafikbildschirm schnellstmöglich mit einem bestimmten Zeichen füllt oder ... mit den Erläuterungen im Listing dürften die Grenzen der Anwendbarkeit dieser Routine nicht so schnell zu erreichen sein.

Maschinensprachroutine 2 zeigt Ihnen, wie Sie einen Bereich im VRAM innerhalb von Sekundenbruchteilen ins RAM kopieren können. In unserem Beispiel wird gar so weit gegangen, daß der gesamte VRAM ins RAM kopiert wird. Ist dies geschehen, können Sie ruhig auf den Textbildschirm zurückkehren, denn ...

mit Maschinenroutine 3 können Sie nun den im RAM soeben abgespeicherten Bildschirm ins VRAM zurückzuholen. So könnten Sie, wie in unserem Beispiel gezeigt, eine Grafik entwickeln, dieselbige mit Maschinenroutine 2 im RAM ablegen, eine zweite Zeichnung erstellen und zwischendurch die bereits abgespeicherte Grafik wieder zurückrufen. Denken Sie daran, daß Ihr RAM-Speicher begrenzt ist! Kopieren Sie nur Teile Ihrer Grafik, dann sind auch hier die Anwendungsmöglichkeiten unbegrenzt.

Maschinenroutine 1   Teil 1

```
10 REM Demonstration des Gebrauchs von
       Maschinenroutinen im MSX-ROM
20 REM 1. &H0056
          Füllen eines Speicherab-
          schnittes im VRAM mit einem
          Zeichen
30 CLEAR 200,39999!
40 SCREEN 0
50 COLOR 1,15
60 WIDTH 40
70 FOR N=40000! TO 40011!
80 READ A$
90 POKE N,VAL("&H"+A$)
100 NEXT N
110 REM &H4D ist die Zahl für das
        Zeichen (Dezimal=77), den
        Buchstaben 'M'
120 DATA 3E,4D
130 REM &H00 und &H00 geben den Start
        zum Ausfüllen im VRAM an
        (Dezimal=0)
140 DATA 21,00,00
150 REM &HC0 und &H03 geben die Länge
        des Speicherbereichs an, der
        mit dem Zeichen ausgefüllt
        werden soll (Dezimal=960)
160 DATA 01,C0,03
170 REM &H56 und &H00 ist die Start-
        adresse der Maschinenroutine
        im ROM zum Ausfüllen eines
        vorher angegebenen Speicher-
        bereichs (Dezimal=86)
180 DATA CD,56,00
190 REM Zurück ins BASIC
200 DATA C9
210 DEFUSR=40000!
220 A=USR(0)
```

Maschinenroutine 2   Teil 1

```
10 REM Demonstration des Gebrauchs von
       Maschinenroutinen im MSX-ROM
20 REM 2. &H0059
          Kopieren eines Speicher
          bereichs aus dem VRAM ins
          RAM
30 CLEAR 200,39999!
40 SCREEN 0
50 COLOR 1,15
60 WIDTH 40
70 FOR N=40000! TO 40012!
80 READ A$
90 POKE N,VAL("&H"+A$)
100 NEXT N
110 REM &H00 und &H00 geben die
        erste Adresse im VRAM an,
        die ins RAM übertragen
        werden soll (Dezimal=0)
120 DATA 21,00,00 'HL
130 REM &H00 und &H40 geben die
        Länge der Speicherplätze an,
        die vom VRAM ins RAM über-
        traen werden sollen
        (Dezimal=16384)
140 DATA 01,00,40 'BC
150 REM &H28 und &HA0 geben die
        erste Adresse im RAM an,
        in die die Werte aus dem
        VRAM fließen sollen
        (Dezimal=41000)
160 DATA 11,28,A0 'DE
170 REM &H59 und &H00 geben die
        Maschinenroutine im ROM an,
        die uns bei der übertragung
        der Daten hilft (Dez.=89)
180 DATA CD,59,00 'Call
190 REM Rückkehr ins BASIC
200 DATA C9
210 DEFUSR=40000!
220 A=USR(0)
```

Maschinenroutine 3 Teil 1

```
220 REM 3. Erstellen einer Zeichnung
        und Abspeicherung im RAM.
230 REM Entwicklung einer Rückhol-
        routine, die uns die Daten
        aus dem RAM wieder ins VRAM
        zurücktransportiert
240 SCREEN 2
250 FOR N=10 TO 80 STEP 5
260 CIRCLE(128,96),N
270 NEXT N
280 FOR N=132 TO 255 STEP 10
290 PAINT(N,96)
300 NEXT N
310 A=USR(0)
320 CLS
330 FOR N=40000! TO 40012!
340 READ A$
350 POKE N,VAL("&H"+A$)
360 NEXT N
370 REM &H28 und &HA0 geben die
        erste Adresse im RAM an,
        die ins VRAM übertragen
        werden soll (Dezimal=41000)
380 DATA 21,28,A0 'HL
390 REM &H00 und &H40 geben die
        Länge der Speicherplätze an,
        die vom RAM ins VRAM über-
        traen werden sollen
        (Dezimal=16384)
400 DATA 01,00,40 'BC
410 REM &H00 und &H00 geben die
        erste Adresse im VRAM an,
        in die die Werte aus dem
        RAM fließen sollen
        (Dezimal=0)
420 DATA 11,00,00 'DE
430 REM &H5C und &H00 geben die
```

Maschinenroutine 3   Teil 2
=================

```
          Maschinenroutine im ROM an,
          die uns bei der übertragung
          der Daten hilft (Dez.=92)
440 DATA CD,5C,00 'Call
450 REM Rückkehr ins BASIC
460 DATA C9
470 DEFUSR=40000!
480 CLS
490 FOR N=255 TO 10 STEP -10
500 LINE(0,0)-(N,N),INT(RND(1)*14+1),BF
510 NEXT N
520 A=USR(0)
530 GOTO 530
```

Wichtige Adressen aus RAM und ROM der MSX-Computer

Hinweis:

Berechnung eines Doppelbytes z.B. 'A und B':
B * 256 + A

&H1BBF bis &H1BBF+2047

Zeichensatz, der beim Computerstart in das VRAM kopiert wird (wie im Anhang auch abgebildet). Diese Adresse befindet sich zudem in den Speicherstellen &HF920 und &HF921.

&HF3AE

An dieser Speicherstelle ist die zur Zeit gültige Bildschirmweite (Zeichen pro Zeile) von SCREEN 0 abgelegt (beim Einschalten = 37). Wenn Sie diesen Wert ändern, können Sie den Vorgang erst durch einen SCREEN 0-Befehl am Bildschirm erkennen.

&HF3AF

An dieser Speicherstelle ist die zur Zeit gültige Bildschirmweite (Zeichen pro Zeile) von SCREEN 1 abgelegt (beim Einschalten = 29). Wenn Sie diesen Wert ändern, können Sie den Vorgang erst durch einen SCREEN 1-Befehl am Bildschirm erkennen.

&HF3B0

An dieser Speicherstelle ist die zur Zeit gültige Bildschirmweite (Zeichen pro Zeile) des SCREENs abgelegt, auf dem Sie sich zur Zeit befinden. Wenn Sie diesen Wert ändern, tritt der eingegebene Zustand (z.B. '30'=30 Zeichen/Zeile) bereits eine Zeile unterhalb der aktuellen Cursorposition in Kraft. Im Gegensatz zum WIDTH-Kommando wird dabei der Bildschirm nicht gelöscht.

&HF3B1

An dieser Speicherstelle ist die Anzahl der Zeilen pro Seite abgespeichert (im Normalfall 24). Sie können diesen Wert nach Belieben ändern, wobei der Wert durch ein SCREEN-Kommando nicht gelöscht wird. Die Eingabe gilt somit gleichzeitig für beide Textbildschirme.

&HF3B3 und &HF3B4

entspricht BASE(0) (Name table von SCREEN 0)

&HF3B7 und &HF3B8

entspricht BASE(2) (Pattern generator table von SCREEN 0)

&HF3BD und &HF3BE

entspricht BASE(5) (Name table von SCREEN 1)

&HF3BF und &HF3C0

entspricht BASE(6) (Colour Table von SCREEN 1)

&HF3C1 und &HF3C2

entspricht BASE(7) (Pattern generator table von SCREEN 1)

&HF3C3 und &HF3C4

entspricht BASE(8) (Sprite attribute table von SCREEN 1)

&HF3C5 und &HF3C6

entspricht BASE(9) (Sprite pattern table von SCREEN 1)

&HF3C7 und &HF3C8

entspricht BASE(10) (Name table von SCREEN 2)

&HF3C9 und &HF3CA

entspricht BASE(11) (Colour table von SCREEN 2)

&HF3CB und &HF3CC

entspricht BASE(12) (Pattern generator table von SCREEN 2)

&HF3CD und &HF3CE

entspricht BASE(13) (Sprite attribute table von SCREEN 2)

&HF3CF und &HF3D0

entspricht BASE(14) (Sprite pattern table von SCREEN 2)

&HF3D1 und &HF3D2

entspricht BASE(15) (Name table von SCREEN 3)

&HF3D5 und &HF3D6

entspricht BASE(17) (Pattern generator table von SCREEN 3)

&HF3D7 und &HF3D8

entspricht BASE(18) (Sprite attribute table von SCREEN 3)

&HF3D9 und &HF3DA

entspricht BASE(19) (Sprite pattern table von SCREEN 3)

&HF3DB

An dieser Speicherstelle ist eine 0 abgespeichert, wenn Sie zuvor durch den SCREEN-Befehl (dritter Parameter) das Tastatur-Klickgeräusch ausgestellt haben bzw. eine 1, wenn Sie das Geräusch zuvor angestellt haben. Neben der Abfrage können Sie das Tastaturklicken selbstverständlich auch durch POKEn in diese Speicherstelle an- bzw. ausstellen.

&HF3DC

An dieser Speicherstelle ist die zur Zeit aktuelle Y-Position des Cursors (in welcher Zeile) abgespeichert. Sie können diese Speicherstelle ähnlich wie den LOCATE-Befehl zur Cursorpositionierung gebrauchen.

&HF3DD

An dieser Speicherstelle ist die zur Zeit aktuelle X-Position des Cursors (in welcher Spalte) abgespeichert. Sie können diese Speicherstelle ähnlich wie den LOCATE-Befehl zur Cursorpositionierung gebrauchen.

&HF3DE

An dieser Speicherstelle ist eine Zahl abgelegt, die darüber Auskunft gibt, ob die Funktionstasten angezeigt werden (entspricht KEY ON) oder nicht (entspricht KEY OFF). Der Wert der Speicherstelle beträgt 255, wenn Sie Funktionstasten sehen und 0, wenn dies nicht der Fall ist. POKEn Sie den entsprechenden Zahlwert in diese Speicherstelle, so wird die Umwandlung erst deutlich, wenn Sie anschließend mit CLS den Bildschirm löschen.

&HF3DF

Entspricht dem aktuellen Inhalt von Register VDP(0)

&HF3E0

Entspricht dem aktuellen Inhalt von Register VDP(1)

&HF3E1

Entspricht dem aktuellen Inhalt von Register VDP(2)

&HF3E2

Entspricht dem aktuellen Inhalt von Register VDP(3)

&HF3E3

Entspricht dem aktuellen Inhalt von Register VDP(4)

&HF3E4

Entspricht dem aktuellen Inhalt von Register VDP(5)

&HF3E5

Entspricht dem aktuellen Inhalt von Register VDP(6)

&HF3E6

Entspricht dem aktuellen Inhalt von Register VDP(7)

&HF3E7

Entspricht dem aktuellen Inhalt von Register VDP(8)

&HF3E9

An dieser Speicherstelle ist eine Zahl abgelegt, die über die aktuelle Vordergrundfarbe (entspricht der Schreibfarbe) Auskunft gibt. POKEn Sie in diese Speicherstelle einen anderen Zahlwert, so wird die Speicherstelle erst gültig d.h. der Computer liest und interpretiert den Inhalt dieser Speicherstelle, wenn Sie ein SCREEN-Kommando eingeben.

&HF3EA

An dieser Speicherstelle ist eine Zahl abgelegt, die über die aktuelle Hintergrundfarbe Auskunft gibt. POKEn Sie in diese Speicherstelle einen anderen Zahlwert, so wird die Speicherstelle erst gültig d.h. der Computer liest und interpretiert den Inhalt dieser Speicherstelle, wenn Sie ein SCREEN-Kommando eingeben.

&HF3EB

An dieser Speicherstelle ist eine Zahl abgelegt, die über die aktuelle Umrandungsfarbe Auskunft gibt (gilt nur in SCREEN 1, 2 und 3). POKEn Sie in diese Speicherstelle einen anderen Zahlwert, so wird die Speicherstelle erst gültig d.h. der Computer liest und interpretiert den Inhalt dieser Speicherstelle, wenn Sie ein SCREEN-Kommando eingeben.

&HF415

Die Abfrage dieser Speicherstelle kommt dem Befehl LPOS gleich. Hier merkt sich Ihr MSX-Computer, an welcher Stelle sich zur Zeit der Kopf des angeschlossenen Druckers befindet.

&HF417

Die Abfrage dieser Speicherstelle kommt dem eingegebenen Wert im fünften Parameter des SCREEN-Befehls (Drucker MSX ja/nein) gleich. Ist der Wert=0, so muß ein MSX-Drucker angeschlossen sein; ist der Wert nicht 0, so werden die speziellen MSX-Grafikzeichen an den Drucker in der Form von Leerzeichen ausgegeben.

&HF87F bis &HF87F+159

In diesen 160 Speicherstellen ist der aktuelle Inhalt der Funktionstasten abgespeichert. Sie können somit durch PEEK die Belegung der einzelnen Funktionstasten direkt abfragen. Auch ist es möglich den aktuellen Inhalt der Funktionstasten über diese Speicherstellen durch POKE zu verändern. Die Änderung der Funktionstasten wird aber erst sichtbar, wenn Sie die SHIFT-Taste einmal betätigen. Wollen Sie augenblicklich den neuen Inhalt der Funktionstasten anzeigen, müssen Sie die Belegung entsprechend ins Video-RAM POKEn (mit VPOKE). Mit VPEEK können Sie zudem auf den Textbildschirmen die aktuelle Belegung der Funktionstasten ablesen (soweit die Bildschirmweite dazu ausreicht, möglichst viele Zeichen der Funktionstzstenbelegung anzuzeigen).

&HF922 und &HF923

In diesen Speicherstellen ist die zur Zeit gültige Adresse abgespeichert, die sich auf den Start des Name table bezieht. Ähnlich wie mit dem Befehl VDP haben Sie die Möglichkeit, nicht nur diese Adresse abzufragen (entspricht einem BASE-Wert), sondern diese Adresse auch augenblicklich durch POKEn zu ändern. Jedoch informieren Sie sich gut, bevor Sie sich an diese Speicherstellen heranwagen (siehe unter BASE, SCREEN und VDP).

&HF924 und &HF925

In diesen Speicherstellen ist die zur Zeit gültige Adresse abgespeichert, die sich auf den Start des Pattern generator table bezieht. Ähnlich wie mit dem Befehl VDP haben Sie die Möglichkeit, nicht nur diese Adresse abzufragen (entspricht einem BASE-Wert), sondern diese Adresse auch augenblicklich durch POKEn zu ändern. Jedoch informieren Sie sich gut, bevor Sie sich an diese Speicherstellen heranwagen (siehe unter BASE, SCREEN und VDP).

&HF926 und &HF927

Inn diesen Speicherstellen ist die zur Zeit gültige Adresse abgespeichert, die sich auf den Start des Sprite pattern table bezieht. Ähnlich wie mit dem Befehl VDP haben Sie die Möglichkeit, nicht nur diese Adresse abzufragen (entspricht einem BASE-Wert), sondern diese Adresse auch augenblicklich durch POKEn zu ändern. Jedoch informieren Sie sich gut, bevor Sie sich an diese Speicherstellen heranwagen (siehe unter BASE, SCREEN und VDP).

&HF928 und &HF929

An diesen Speicherstellen ist die zur Zeit gültige Adresse abgespeichert, die sich auf den Start des Sprite attribute table bezieht. Ähnlich wie mit dem Befehl VDP haben Sie die Möglichkeit, nicht nur diese Adresse abzufragen (entspricht einem BASE-Wert), sondern diese Adresse auch augenblicklich durch POKEn zu ändern. Jedoch informieren Sie sich gut, bevor Sie sich an diese Speicherstellen heranwagen (siehe unter BASE, SCREEN und VDP).

&HF956 und folgende

Hier befindet sich der Speicherraum, den der PLAY-Befehl benötigt, um sich die eingegebenen Noten zu merken. Von hier aus wird per Interruptroutine Ton für Ton von Ihrem MSX-Computer eingelesen. Zur genaueren Beschreibung einiger Speicherplätze in diesem Speicherraum siehe unter PLAY(N).

&HFBCC

Sie wissen, daß sich der Cursor in seinem Aussehen ändert, wenn Sie ihn mit den Cursortasten auf ein anderes Zeichen dirigieren. In dieser Speicherstelle ist das aktuelle Zeichen abgespeichert, daß sich unter dem Cursor zur Zeit befindet (dieses Zeichen wird invers dargestellt).

&HFC9D bzw. &HFC9C

Hier sind X- und Y-Koordinate des angeschlossenen Pads (abfragbar durch den MSX-BASIC-Befehl PAD) abgespeichert.

&HFCA9

Der Wert dieser Speicherstelle entspricht dem Wert, den Sie mit dem dritten LOCATE-Parameter eingeben (soll der Cursor während des Programmablaufs auf dem Bildschirm erscheinen? ja=1 nein=0). Sie können somit den LOCATE-Parameter auch über diese Speicherstelle ändern.

&HFCAB

In dieser Speicherstelle steht eine 0, wenn Sie mit Klein- und Großbuchstaben arbeiten wollen (die Großbuchstaben werden durch das gleichzeitige Drücken der SHIFT-Taste angesteuert) bzw. eine 1, wenn Sie nur mit Großbuchstaben arbeiten wollen. Geben Sie durch POKE eine 1 in diese Speicherstelle ein, schreiben Sie von nun an in Großschrift, ohne die CAPS-Taste betätigt zu haben (die LED in der CAPS-Taste leuchtet allerdings nicht auf).

&HFCAF

Hier ist die Zahl des aktuellen SCREENs abgespeichert.

&HFCB0

Hier ist die Zahl des SCREENs abgespeichert, der zuletzt angewählt wurde. Somit ist es möglich, über diese Speicherstelle zu erfahren, auf welchem Textbildschirm wir uns zuletzt aufgehalten haben, während wir nun auf einem Grafikbildschirm arbeiten.

&HFCB2

Sie haben in SCREEN 3 die Möglichkeit, bei Gebrauch des PAINT-Befehls nicht nur die Ausmalfarbe sondern gleichzeitig auch die Farbe einzugeben, bis zu der ausgemalt werden soll. In der hier besprochenen Speicherstelle wird diese Information festgehalten.

&HFCB3 bzw. &HFCB5 bzw. &HFCB7 bzw. &HFCB9

In den ersten zwei Speicherstellen ist die Information über die aktuellen X-Y-Koordinaten des Grafikcursors festgehalten. In den beiden weiteren Speicherstellen ist die Information darüber abgespeichert, inwieweit und ob sich ein relativer Grafikcursor durch einen relativen Befehl (DRAW mit 'N', STEP-Befehl, Entfernung der Punkte beim Kreis vom Kreismittelpunkt) vom absoluten Grafikcursor unterscheidet.

&HFCBC

Sie können im DRAW-Befehl unterschiedliche Maßstäbe durch Eingabe des 'S'-Parameters und eines Zahlwertes ansteuern. Die aktuelle Vergrößerung/Verkleinerung (4 ist der Urzustand und bedeutet Maßstab Eingabe:Anzeige = 1:1) wird in dieser Speicherstelle festgehalten und Sie können diesen Maßstab auch selbstverständlich durch POKE ändern.

&HFCBD

Sie können im DRAW-Befehl unterschiedliche Winkel im 90 Grad-Rhythmus durch Eingabe des 'A'-Parameters und eines Zahlwertes ansteuern. Die aktuelle Drehung (0 = keine Drehung ist der Urzustand) wird in dieser Speicherstelle festgehalten und Sie können diesen Winkel auch selbstverständlich durch POKE ändern.

Mögliche Werte, die wir für VDP-Register 2 eingeben können

00	0000	01	0400	02	0800	03	0C00
04	1000	05	1400	06	1800	07	1C00
08	2000	09	2400	0A	2800	0B	2C00
0C	3000	0D	3400	0E	3800	0F	3C00

Mögliche Werte, die wir für VDP-Register 4 eingeben können

00	0000	01	0800	02	1000	03	1800
04	2000	05	2800	06	3000	07	3800

Mögliche Werte, die wir für VDP-Register 5 eingeben können

00	0000	01	0080	02	0100	03	0180
04	0200	05	0280	06	0300	07	0380
08	0400	09	0480	0A	0500	0B	0580
0C	0600	0D	0680	0E	0700	0F	0780
10	0800	11	0880	12	0900	13	0980
14	0A00	15	0A80	16	0B00	17	0B80
18	0C00	19	0C80	1A	0D00	1B	0D80
1C	0E00	1D	0E80	1E	0F00	1F	0F80
20	1000	21	1080	22	1100	23	1180
24	1200	25	1280	26	1300	27	1380
28	1400	29	1480	2A	1500	2B	1580
2C	1600	2D	1680	2E	1700	2F	1780
30	1800	31	1880	32	1900	33	1980
34	1A00	35	1A80	36	1B00	37	1B80
38	1C00	39	1C80	3A	1D00	3B	1D80
3C	1E00	3D	1E80	3E	1F00	3F	1F80
40	2000	41	2080	42	2100	43	2180
44	2200	45	2280	46	2300	47	2380
48	2400	49	2480	4A	2500	4B	2580
4C	2600	4D	2680	4E	2700	4F	2780
50	2800	51	2880	52	2900	53	2980
54	2A00	55	2A80	56	2B00	57	2B80
58	2C00	59	2C80	5A	2D00	5B	2D80
5C	2E00	5D	2E80	5E	2F00	5F	2F80
60	3000	61	3080	62	3100	63	3180
64	3200	65	3280	66	3300	67	3380
68	3400	69	3480	6A	3500	6B	3580
6C	3600	6D	3680	6E	3700	6F	3780
70	3800	71	3880	72	3900	73	3980
74	3A00	75	3A80	76	3B00	77	3B80
78	3C00	79	3C80	7A	3D00	7B	3D80
7C	3E00	7D	3E80	7E	3F00	7F	3F80

Mögliche Werte, die wir für VDP-Register 6 eingeben können

00	0000	01	0800	02	1000	03	1800
04	2000	05	2800	06	3000	07	3800

Mögliche Werte, die wir für VDP-Register 3 eingeben können
===

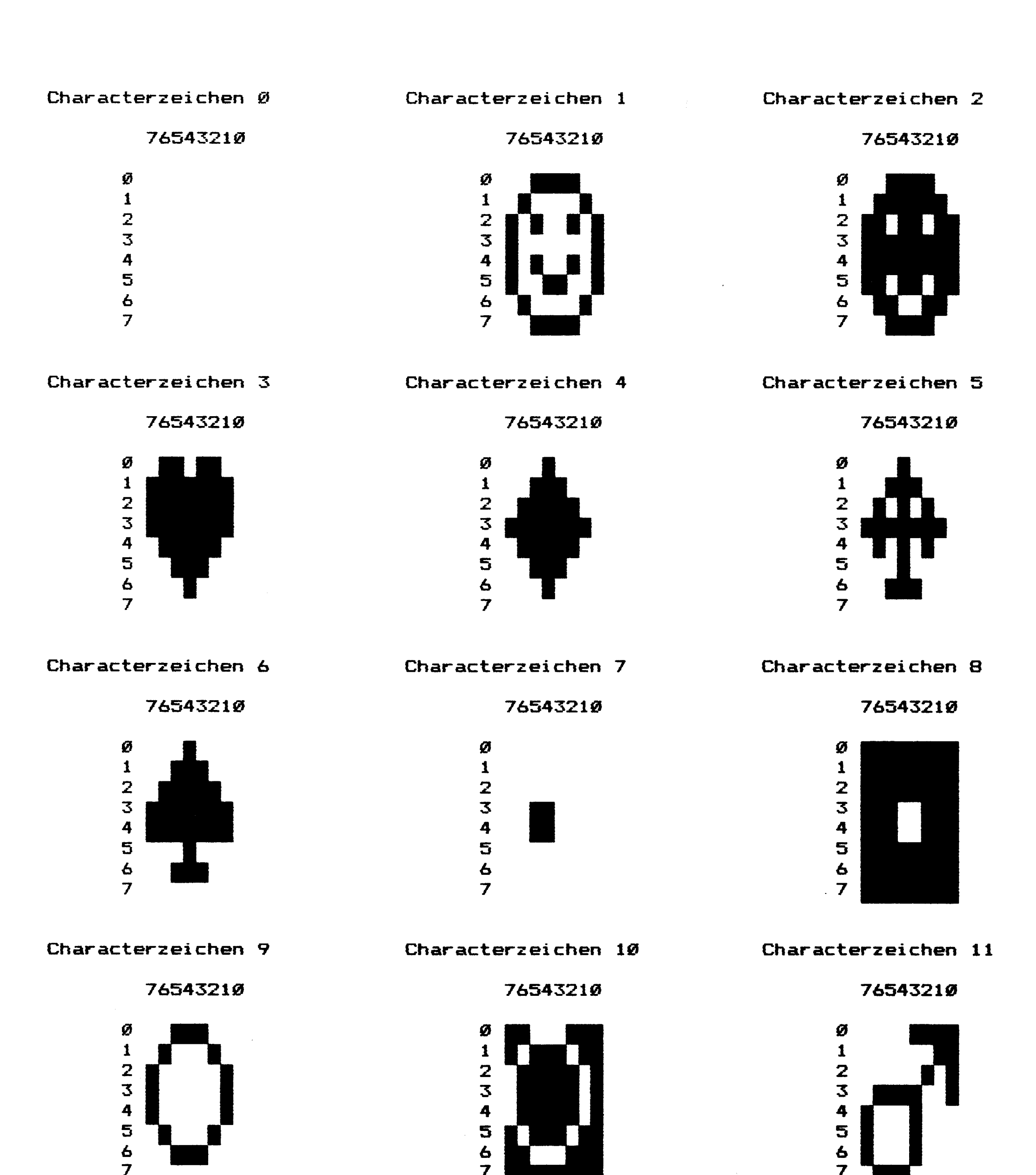
Teil 1

00	0000	01	0040	02	0080	03	00C0
04	0100	05	0140	06	0180	07	01C0
08	0200	09	0240	0A	0280	0B	02C0
0C	0300	0D	0340	0E	0380	0F	03C0
10	0400	11	0440	12	0480	13	04C0
14	0500	15	0540	16	0580	17	05C0
18	0600	19	0640	1A	0680	1B	06C0
1C	0700	1D	0740	1E	0780	1F	07C0
20	0800	21	0840	22	0880	23	08C0
24	0900	25	0940	26	0980	27	09C0
28	0A00	29	0A40	2A	0A80	2B	0AC0
2C	0B00	2D	0B40	2E	0B80	2F	0BC0
30	0C00	31	0C40	32	0C80	33	0CC0
34	0D00	35	0D40	36	0D80	37	0DC0
38	0E00	39	0E40	3A	0E80	3B	0EC0
3C	0F00	3D	0F40	3E	0F80	3F	0FC0
40	1000	41	1040	42	1080	43	10C0
44	1100	45	1140	46	1180	47	11C0
48	1200	49	1240	4A	1280	4B	12C0
4C	1300	4D	1340	4E	1380	4F	13C0
50	1400	51	1440	52	1480	53	14C0
54	1500	55	1540	56	1580	57	15C0
58	1600	59	1640	5A	1680	5B	16C0
5C	1700	5D	1740	5E	1780	5F	17C0
60	1800	61	1840	62	1880	63	18C0
64	1900	65	1940	66	1980	67	19C0
68	1A00	69	1A40	6A	1A80	6B	1AC0
6C	1B00	6D	1B40	6E	1B80	6F	1BC0
70	1C00	71	1C40	72	1C80	73	1CC0
74	1D00	75	1D40	76	1D80	77	1DC0
78	1E00	79	1E40	7A	1E80	7B	1EC0
7C	1F00	7D	1F40	7E	1F80	7F	1FC0

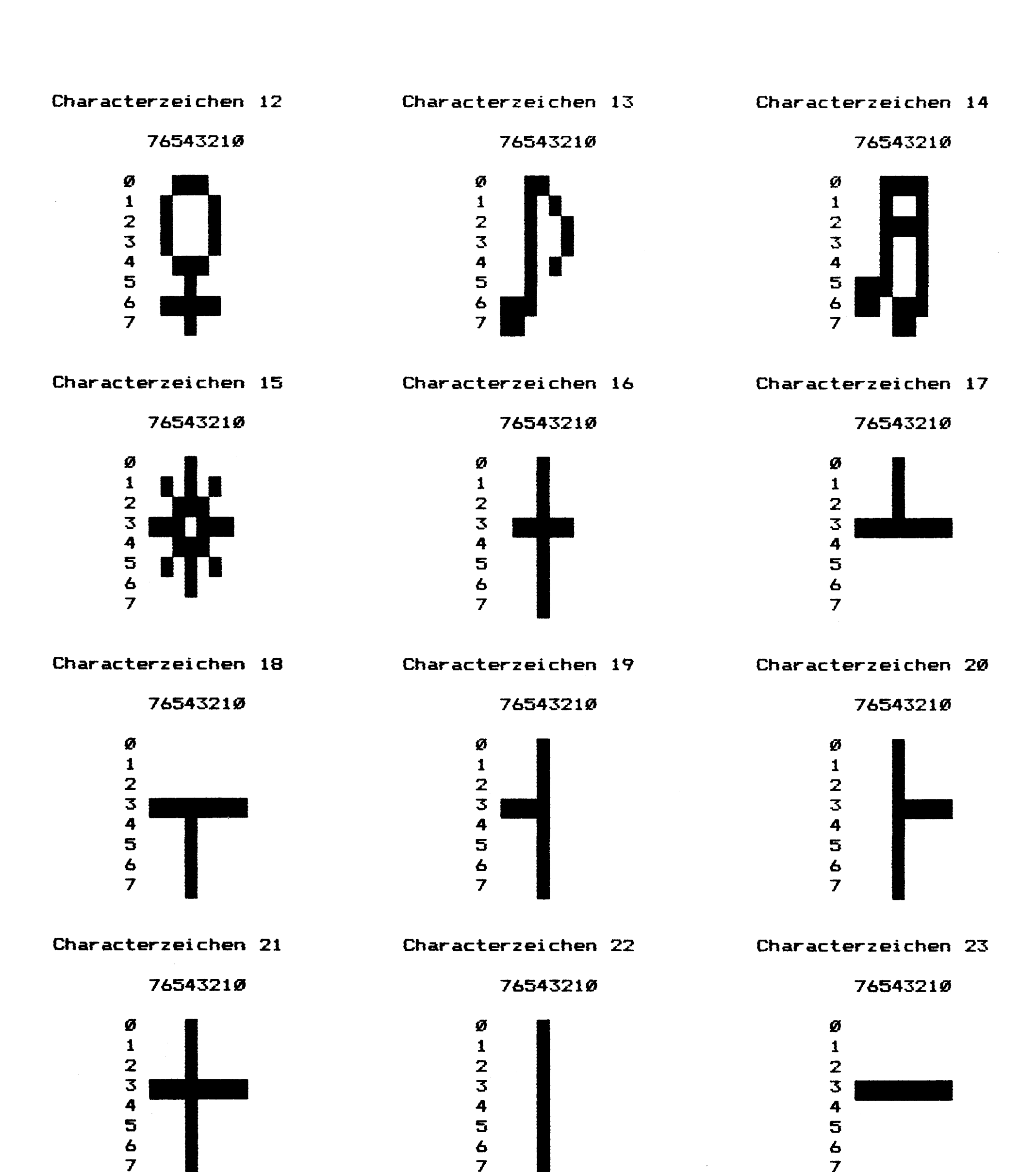
Mögliche Werte, die wir für VDP-Register 3 eingeben können
===

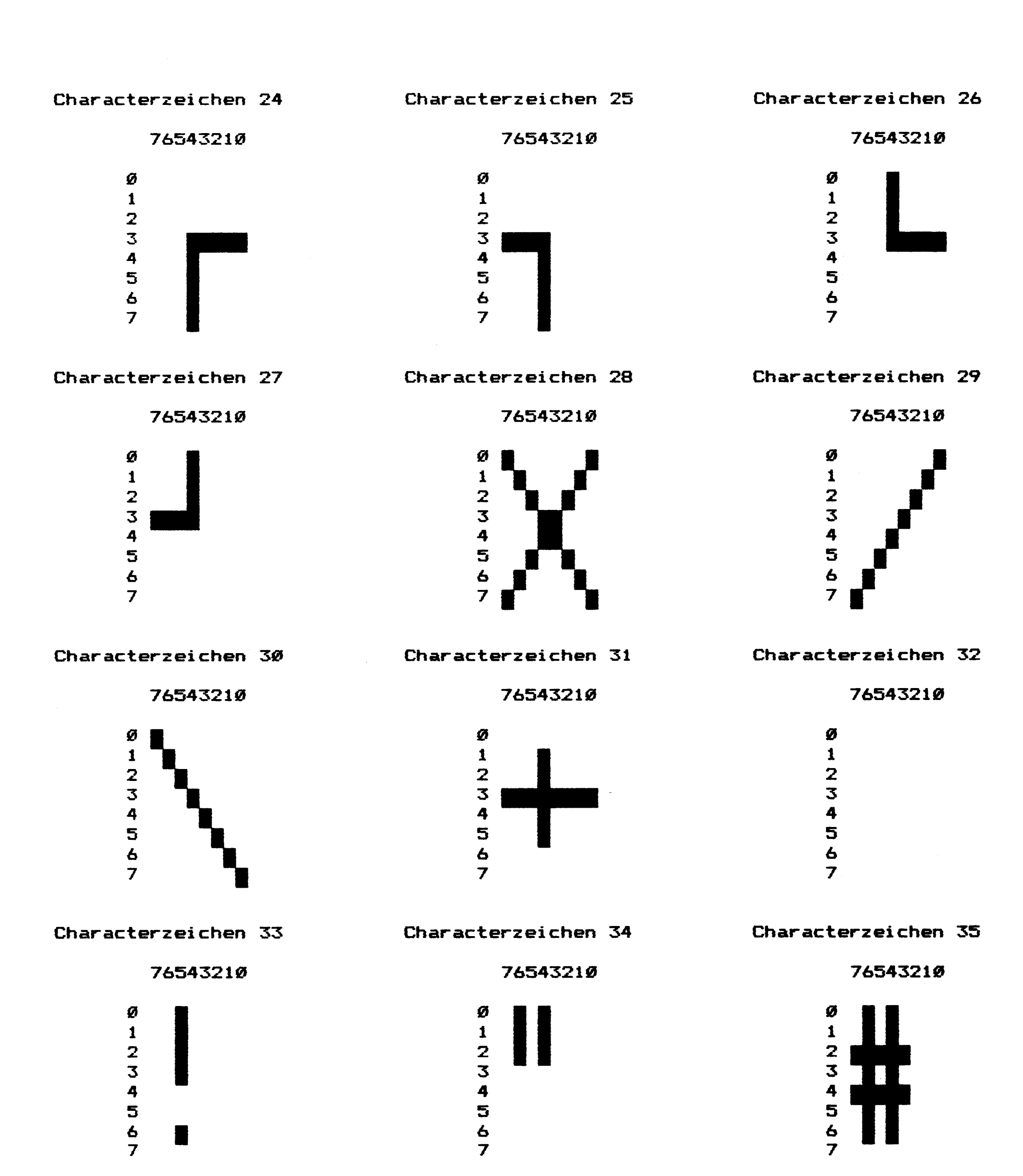
Teil 2

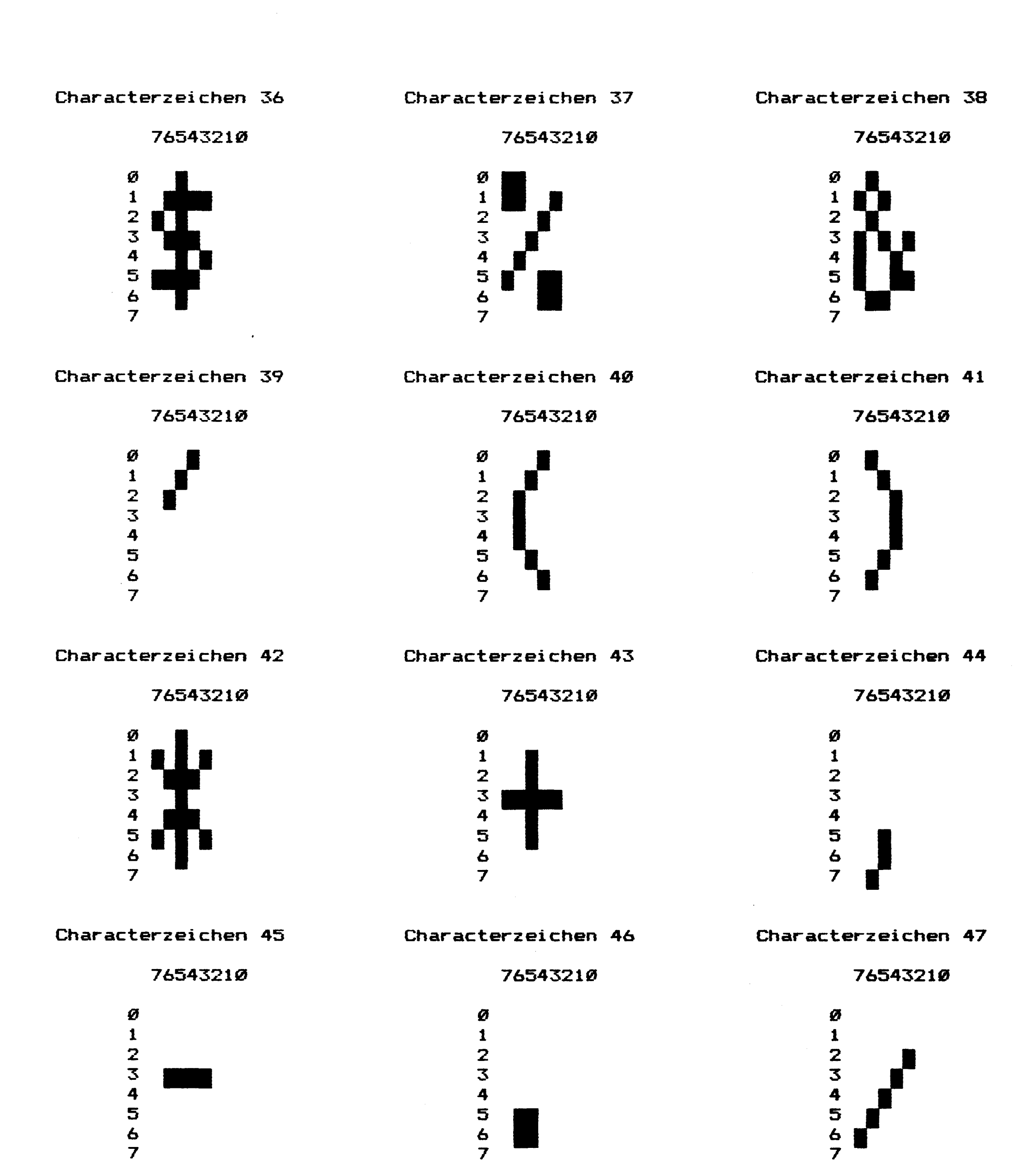
80	2000	81	2040	82	2080	83	20C0
84	2100	85	2140	86	2180	87	21C0
88	2200	89	2240	8A	2280	8B	22C0
8C	2300	8D	2340	8E	2380	8F	23C0
90	2400	91	2440	92	2480	93	24C0
94	2500	95	2540	96	2580	97	25C0
98	2600	99	2640	9A	2680	9B	26C0
9C	2700	9D	2740	9E	2780	9F	27C0
A0	2800	A1	2840	A2	2880	A3	28C0
A4	2900	A5	2940	A6	2980	A7	29C0
A8	2A00	A9	2A40	AA	2A80	AB	2AC0
AC	2B00	AD	2B40	AE	2B80	AF	2BC0
B0	2C00	B1	2C40	B2	2C80	B3	2CC0
B4	2D00	B5	2D40	B6	2D80	B7	2DC0
B8	2E00	B9	2E40	BA	2E80	BB	2EC0
BC	2F00	BD	2F40	BE	2F80	BF	2FC0
C0	3000	C1	3040	C2	3080	C3	30C0
C4	3100	C5	3140	C6	3180	C7	31C0
C8	3200	C9	3240	CA	3280	CB	32C0
CC	3300	CD	3340	CE	3380	CF	33C0
D0	3400	D1	3440	D2	3480	D3	34C0
D4	3500	D5	3540	D6	3580	D7	35C0
D8	3600	D9	3640	DA	3680	DB	36C0
DC	3700	DD	3740	DE	3780	DF	37C0
E0	3800	E1	3840	E2	3880	E3	38C0
E4	3900	E5	3940	E6	3980	E7	39C0
E8	3A00	E9	3A40	EA	3A80	EB	3AC0
EC	3B00	ED	3B40	EE	3B80	EF	3BC0
F0	3C00	F1	3C40	F2	3C80	F3	3CC0
F4	3D00	F5	3D40	F6	3D80	F7	3DC0
F8	3E00	F9	3E40	FA	3E80	FB	3EC0
FC	3F00	FD	3F40	FE	3F80	FF	3FC0

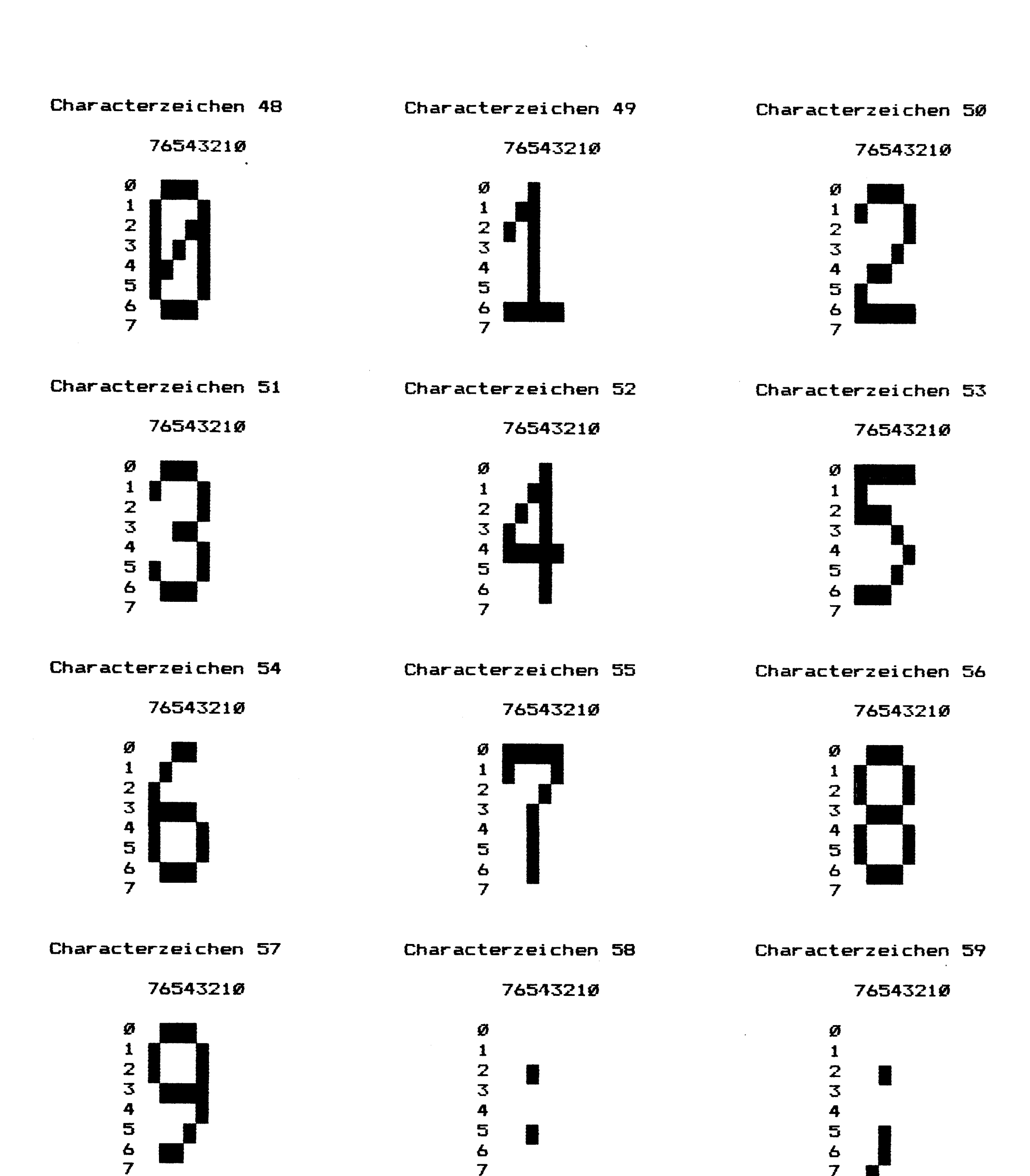


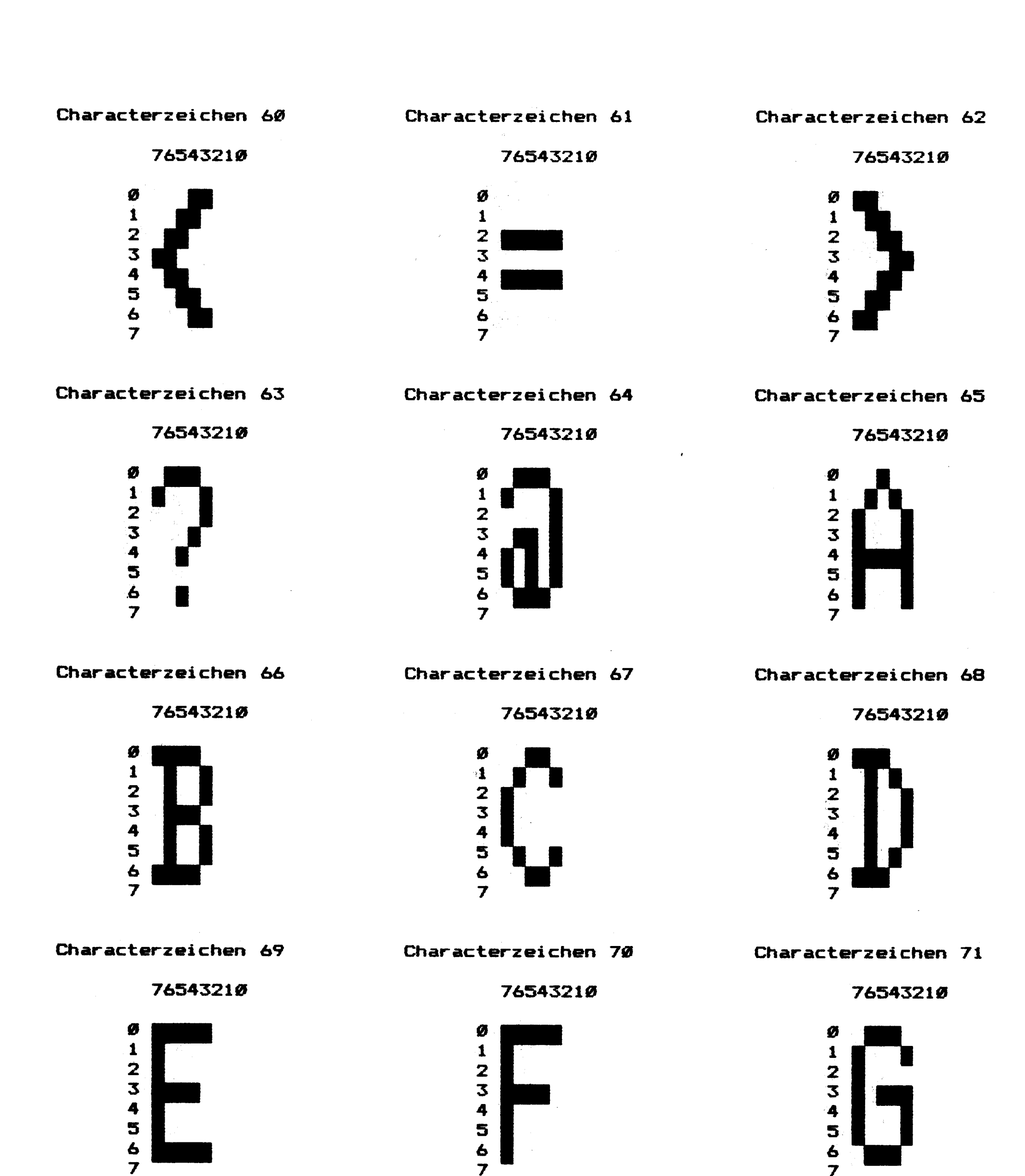
Characterzeichen 0
76543210
0 1 2 3 4 5 6 7
Characterzeichen 1
76543210
0 1 2 3 4 5 6 7
Characterzeichen 2
76543210
0 1 2 3 4 5 6 7
Characterzeichen 3
76543210
0 1 2 3 4 5 6 7
Characterzeichen 4
76543210
0 1 2 3 4 5 6 7
Characterzeichen 5
76543210
0 1 2 3 4 5 6 7
Characterzeichen 6
76543210
0 1 2 3 4 5 6 7
Characterzeichen 7
76543210
0 1 2 3 4 5 6 7
Characterzeichen 8
76543210
0 1 2 3 4 5 6 7
Characterzeichen 9
76543210
0 1 2 3 4 5 6 7
Characterzeichen 10
76543210
0 1 2 3 4 5 6 7
Characterzeichen 11
76543210
0 1 2 3 4 5 6 7


Characterzeichen 12
76543210
0 1 2 3 4 5 6 7
Characterzeichen 13
76543210
0 1 2 3 4 5 6 7
Characterzeichen 14
76543210
0 1 2 3 4 5 6 7
Characterzeichen 15
76543210
0 1 2 3 4 5 6 7
Characterzeichen 16
76543210
0 1 2 3 4 5 6 7
Characterzeichen 17
76543210
0 1 2 3 4 5 6 7
Characterzeichen 18
76543210
0 1 2 3 4 5 6 7
Characterzeichen 19
76543210
0 1 2 3 4 5 6 7
Characterzeichen 20
76543210
0 1 2 3 4 5 6 7
Characterzeichen 21
76543210
0 1 2 3 4 5 6 7
Characterzeichen 22
76543210
0 1 2 3 4 5 6 7
Characterzeichen 23
76543210
0 1 2 3 4 5 6 7

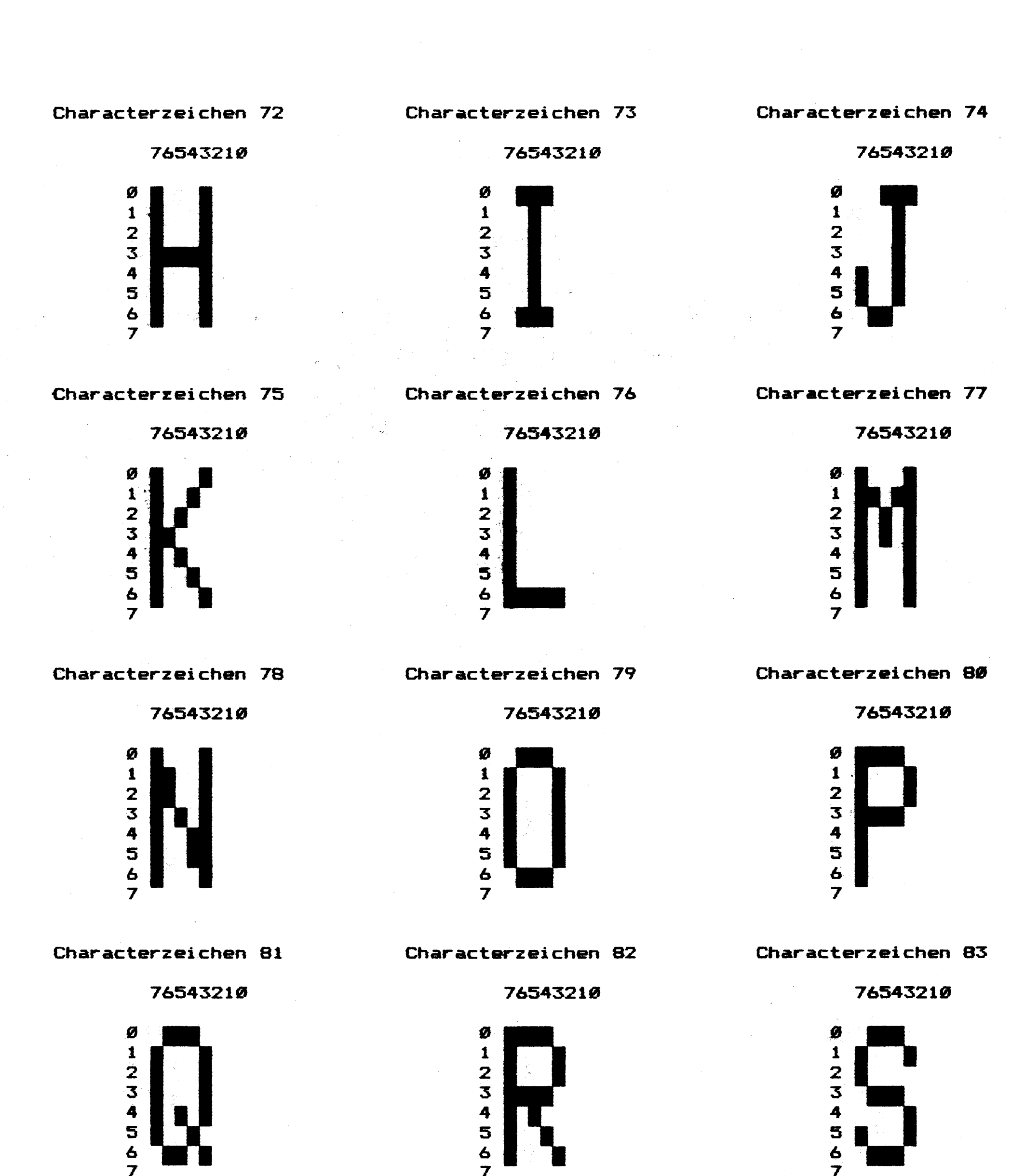

Characterzeichen 24
76543210
0 1 2 3 4 5 6 7
Characterzeichen 25
76543210
0 1 2 3 4 5 6 7
Characterzeichen 26
76543210
0 1 2 3 4 5 6 7
Characterzeichen 27
76543210
0 1 2 3 4 5 6 7
Characterzeichen 28
76543210
0 1 2 3 4 5 6 7
Characterzeichen 29
76543210
0 1 2 3 4 5 6 7
Characterzeichen 30
76543210
0 1 2 3 4 5 6 7
Characterzeichen 31
76543210
0 1 2 3 4 5 6 7
Characterzeichen 32
76543210
0 1 2 3 4 5 6 7
Characterzeichen 33
76543210
0 1 2 3 4 5 6 7
Characterzeichen 34
76543210
0 1 2 3 4 5 6 7
Characterzeichen 35
76543210
0 1 2 3 4 5 6 7

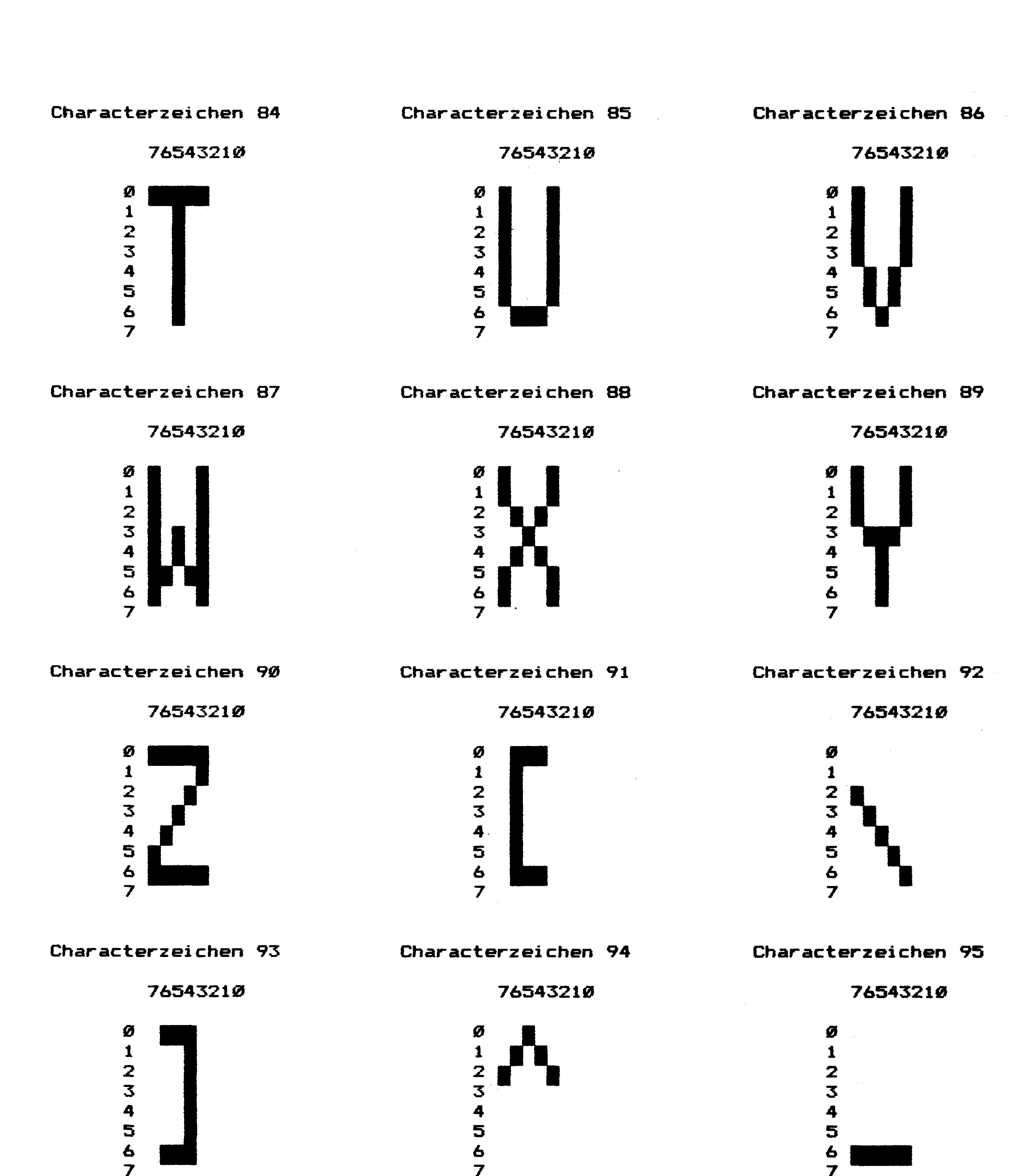

Characterzeichen 36
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 37
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 38
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 39
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 4Ø
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 41
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 42
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 43
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 44
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 45
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 46
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 47
76543210
Ø 1 2 3 4 5 6 7

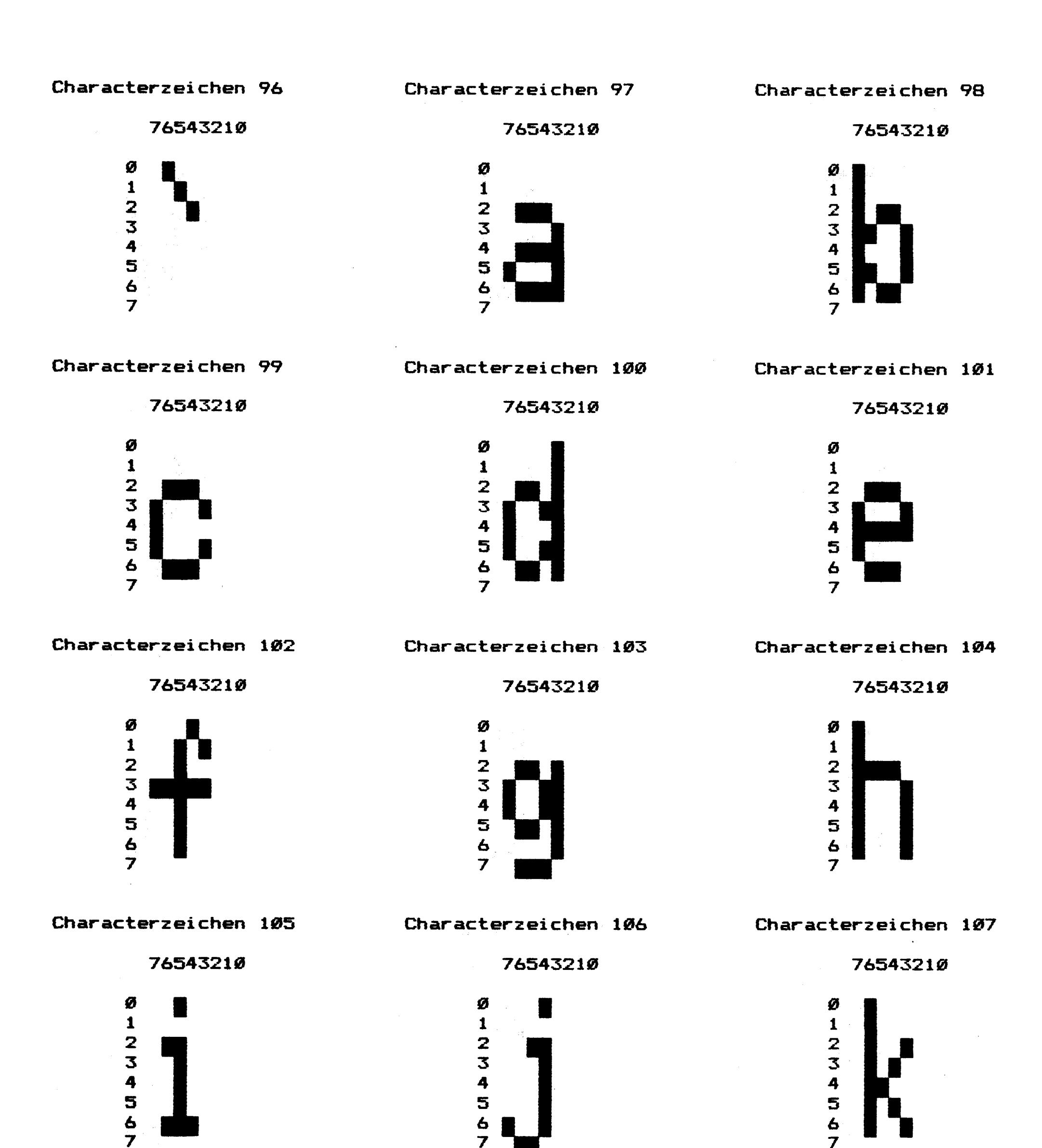

Characterzeichen 48
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 49
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 5Ø
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 51
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 52
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 53
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 54
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 55
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 56
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 57
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 58
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 59
76543210
Ø 1 2 3 4 5 6 7



Characterzeichen 6Ø
7654321Ø
Characterzeichen 61
7654321Ø
Characterzeichen 62
7654321Ø
Characterzeichen 63
7654321Ø
Characterzeichen 64
7654321Ø
Characterzeichen 65
7654321Ø
Characterzeichen 66
7654321Ø
Characterzeichen 67
7654321Ø
Characterzeichen 68
7654321Ø
Characterzeichen 69
7654321Ø
Characterzeichen 7Ø
7654321Ø
Characterzeichen 71
7654321Ø


Characterzeichen 72
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 73
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 74
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 75
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 76
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 77
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 78
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 79
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 8Ø
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 81
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 82
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 83
7654321Ø
Ø 1 2 3 4 5 6 7


Characterzeichen 84
76543210
0
1
2
3
4
5
6
7
Characterzeichen 85
76543210
0
1
2
3
4
5
6
7
Characterzeichen 86
76543210
0
1
2
3
4
5
6
7
Characterzeichen 87
76543210
0
1
2
3
4
5
6
7
Characterzeichen 88
76543210
0
1
2
3
4
5
6
7
Characterzeichen 89
76543210
0
1
2
3
4
5
6
7
Characterzeichen 90
76543210
0
1
2
3
4
5
6
7
Characterzeichen 91
76543210
0
1
2
3
4
5
6
7
Characterzeichen 92
76543210
0
1
2
3
4
5
6
7
Characterzeichen 93
76543210
0
1
2
3
4
5
6
7
Characterzeichen 94
76543210
0
1
2
3
4
5
6
7
Characterzeichen 95
76543210
0
1
2
3
4
5
6
7


Characterzeichen 96
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 97
76543210
Characterzeichen 98
76543210
Characterzeichen 99
76543210
Characterzeichen 100
76543210
Characterzeichen 101
76543210
Characterzeichen 102
76543210
Characterzeichen 103
76543210
Characterzeichen 104
76543210
Characterzeichen 105
76543210
Characterzeichen 106
76543210
Characterzeichen 107
76543210

Characterzeichen 108

76543210

0
1
2
3
4
5
6
7

Characterzeichen 109

76543210

0
1
2
3
4
5
6
7

Characterzeichen 110

76543210

0
1
2
3
4
5
6
7

Characterzeichen 111

76543210

0
1
2
3
4
5
6
7

Characterzeichen 112

76543210

0
1
2
3
4
5
6
7

Characterzeichen 113

76543210

0
1
2
3
4
5
6
7

Characterzeichen 114

76543210

0
1
2
3
4
5
6
7

Characterzeichen 115

76543210

0
1
2
3
4
5
6
7

Characterzeichen 116

76543210

0
1
2
3
4
5
6
7

Characterzeichen 117

76543210

0
1
2
3
4
5
6
7

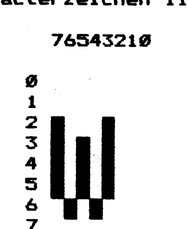
Characterzeichen 118

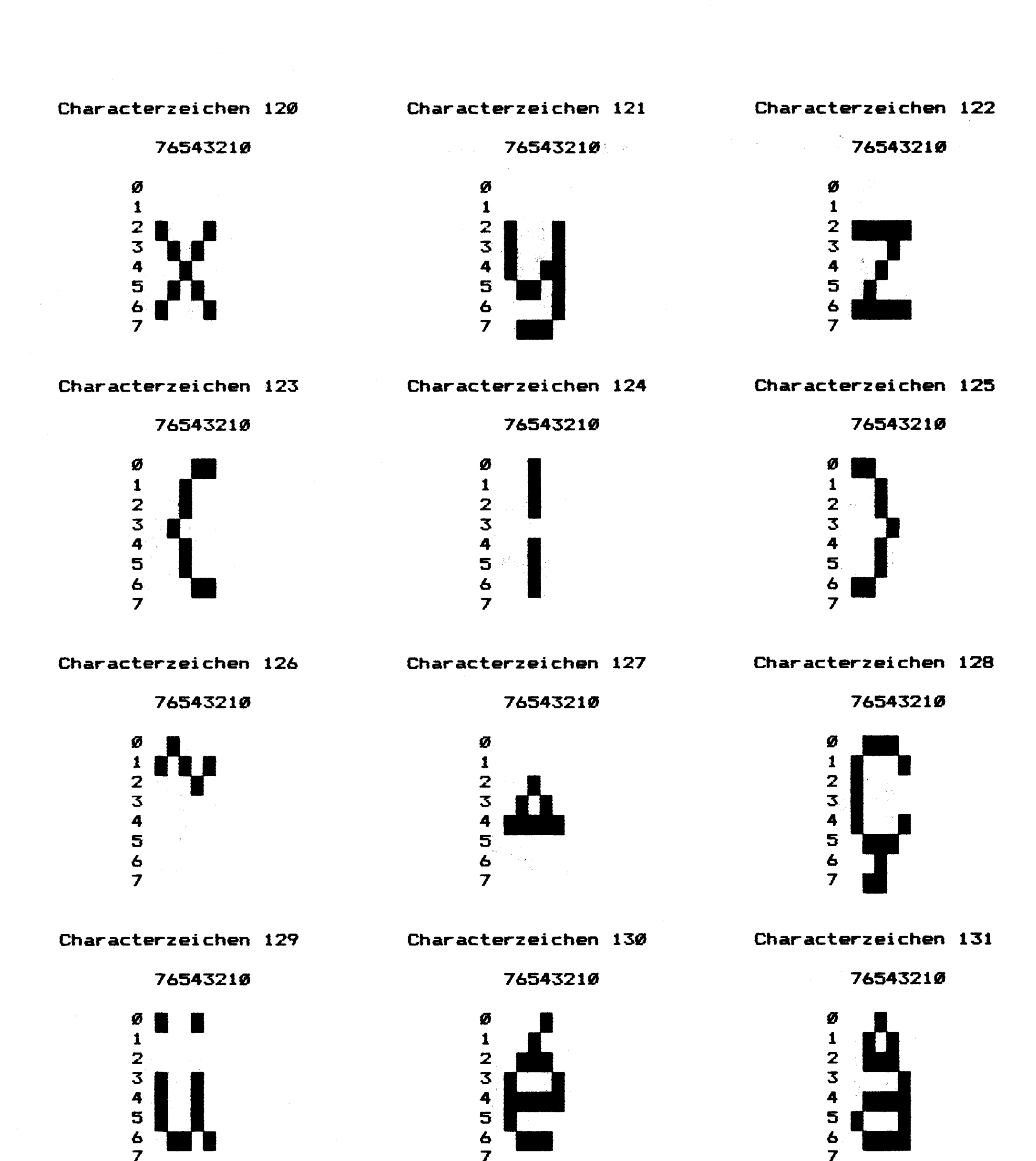
76543210

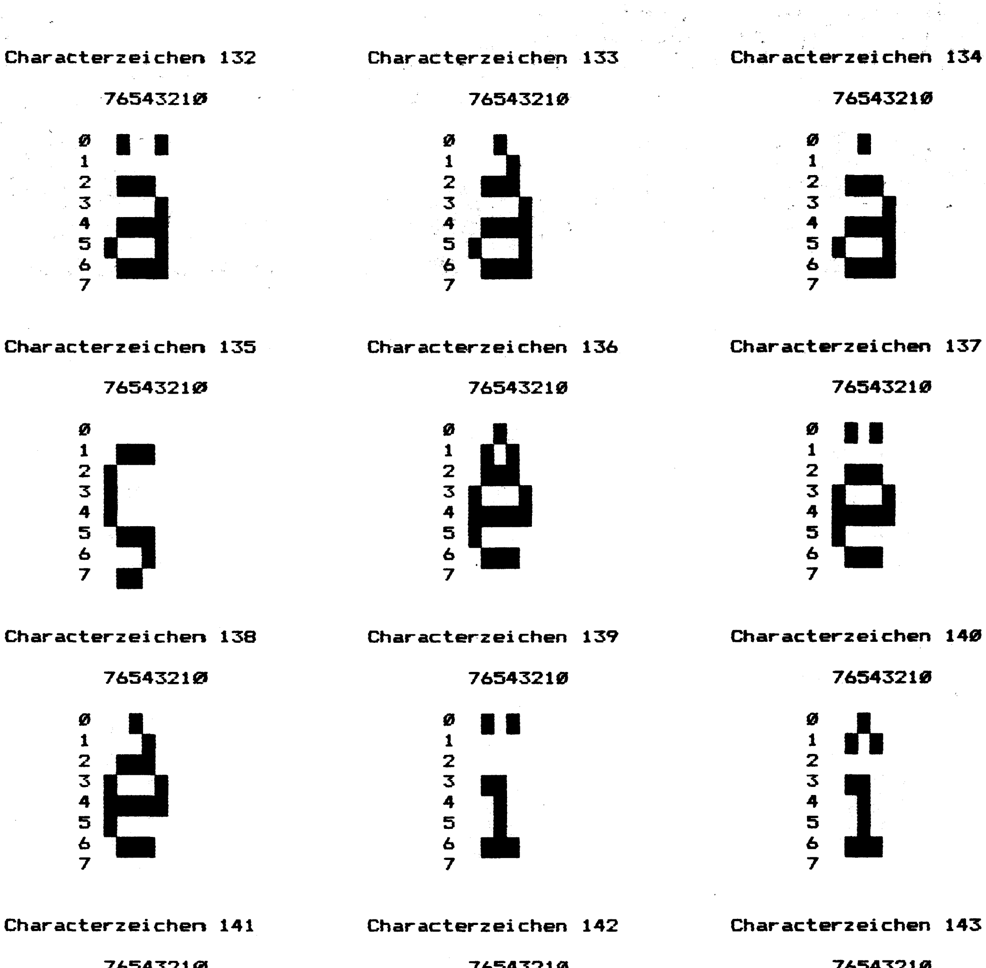
0
1
2
3
4
5
6
7

Characterzeichen 119

76543210




Characterzeichen 120
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 121
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 122
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 123
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 124
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 125
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 126
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 127
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 128
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 129
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 130
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 131
76543210
Ø
1
2
3
4
5
6
7



Characterzeichen 141

76543210

0
1
2
3
4
5
6
7

Characterzeichen 142

76543210

0
1
2
3
4
5
6
7

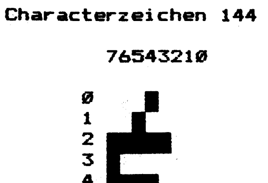
Characterzeichen 143

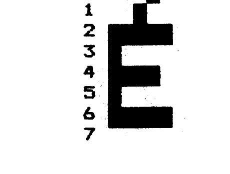
76543210

0
1
2
3
4
5
6
7

Characterzeichen 144

76543210



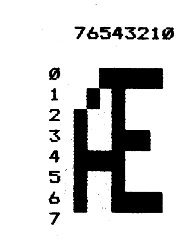


Characterzeichen 145

76543210

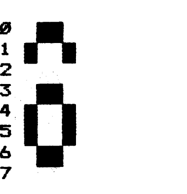
Characterzeichen 146

76543210



Characterzeichen 147

76543210

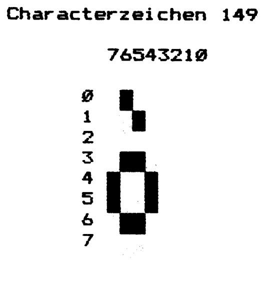


Characterzeichen 148

76543210

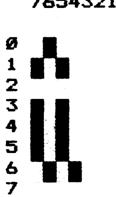
Characterzeichen 149

76543210



Characterzeichen 150

76543210

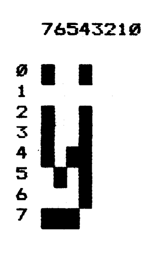


Characterzeichen 151

76543210

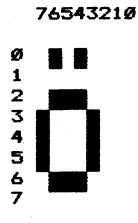
Characterzeichen 152

76543210

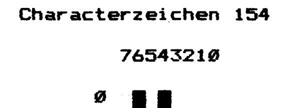


Characterzeichen 153

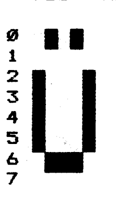
76543210



Characterzeichen 154

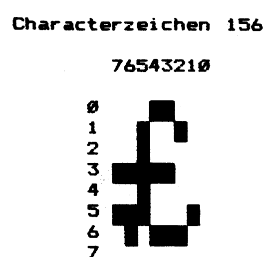


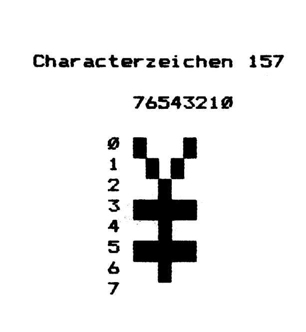
76543210

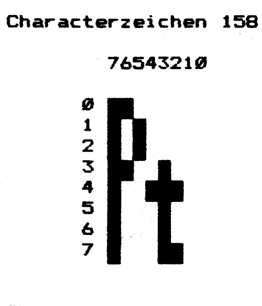


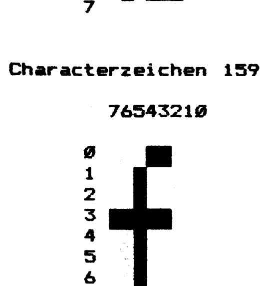
Characterzeichen 155

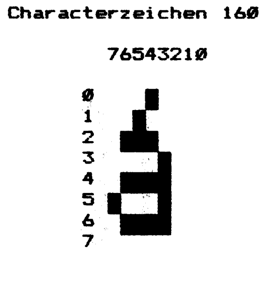
76543210

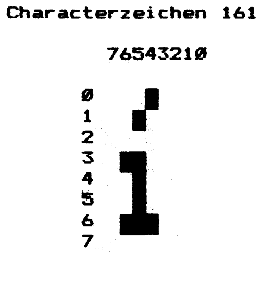

Characterzeichen 156
76543210

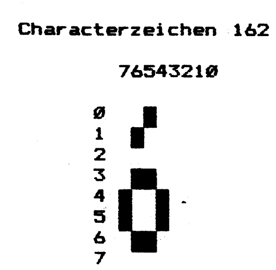

Characterzeichen 157
76543210

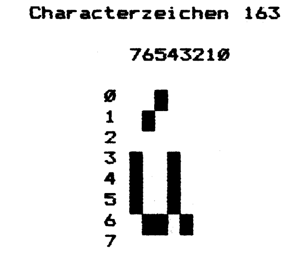

Characterzeichen 158
76543210

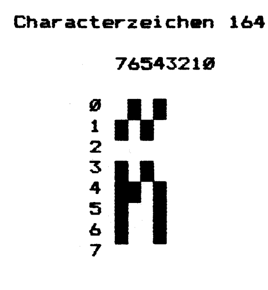

Characterzeichen 159
76543210

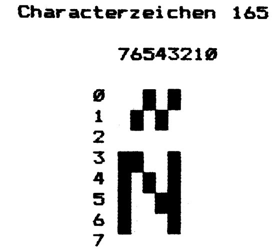

Characterzeichen 160
76543210

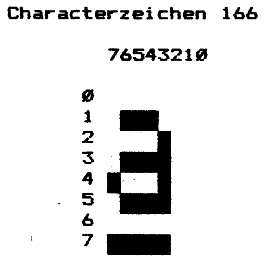

Characterzeichen 161
76543210

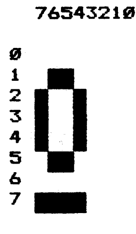

Characterzeichen 162
76543210

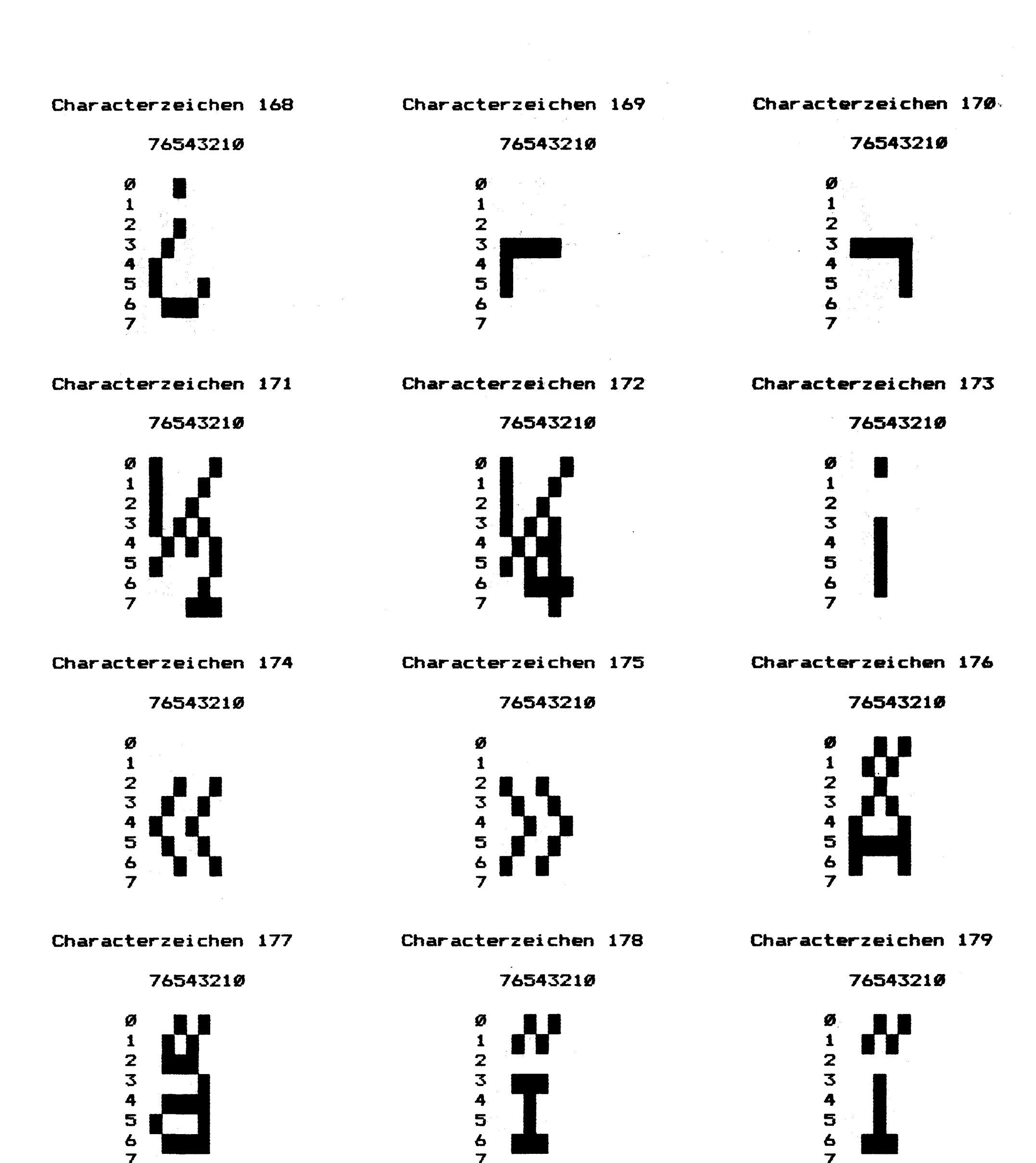

Characterzeichen 163
76543210

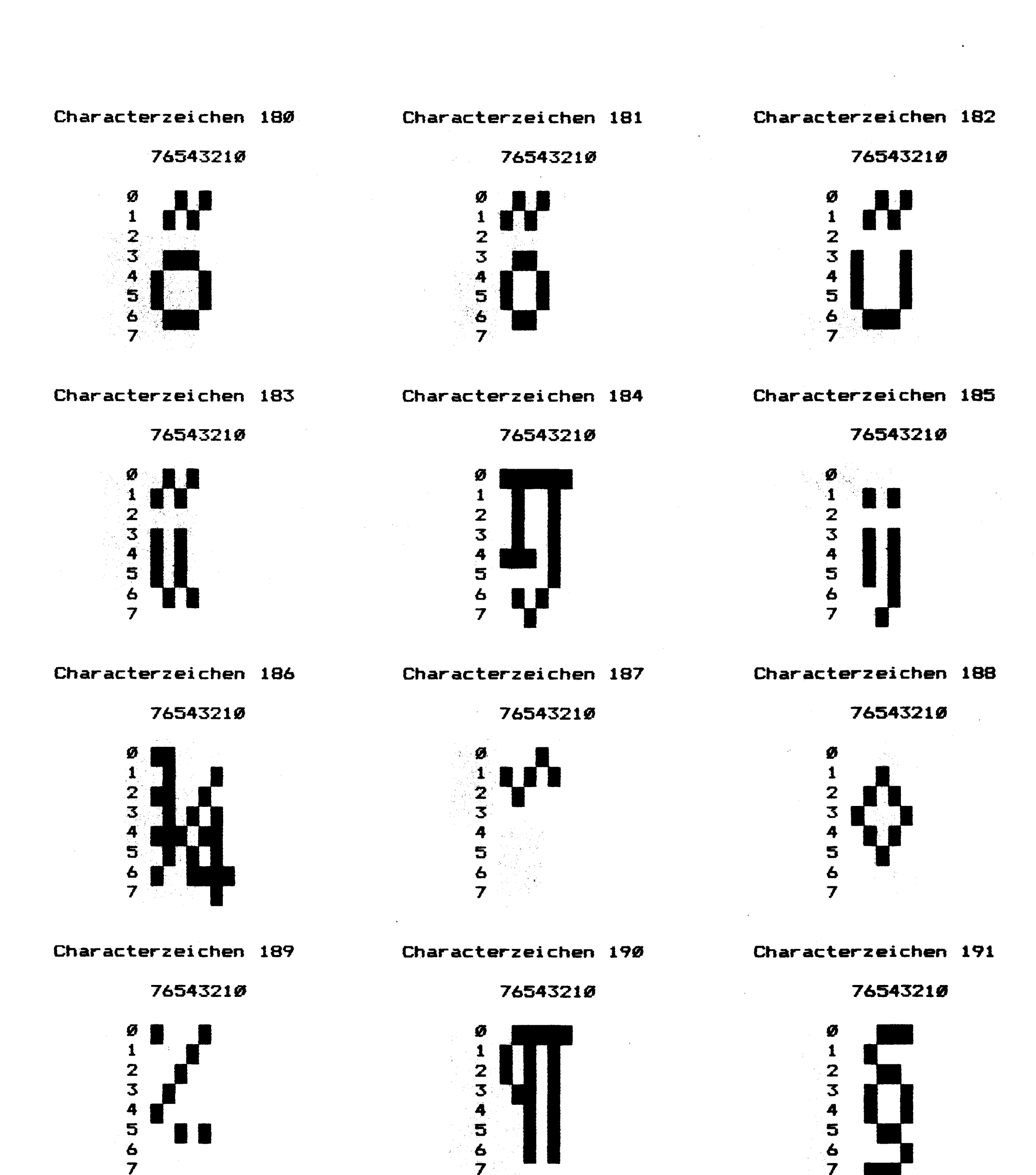

Characterzeichen 164
76543210


Characterzeichen 165
76543210

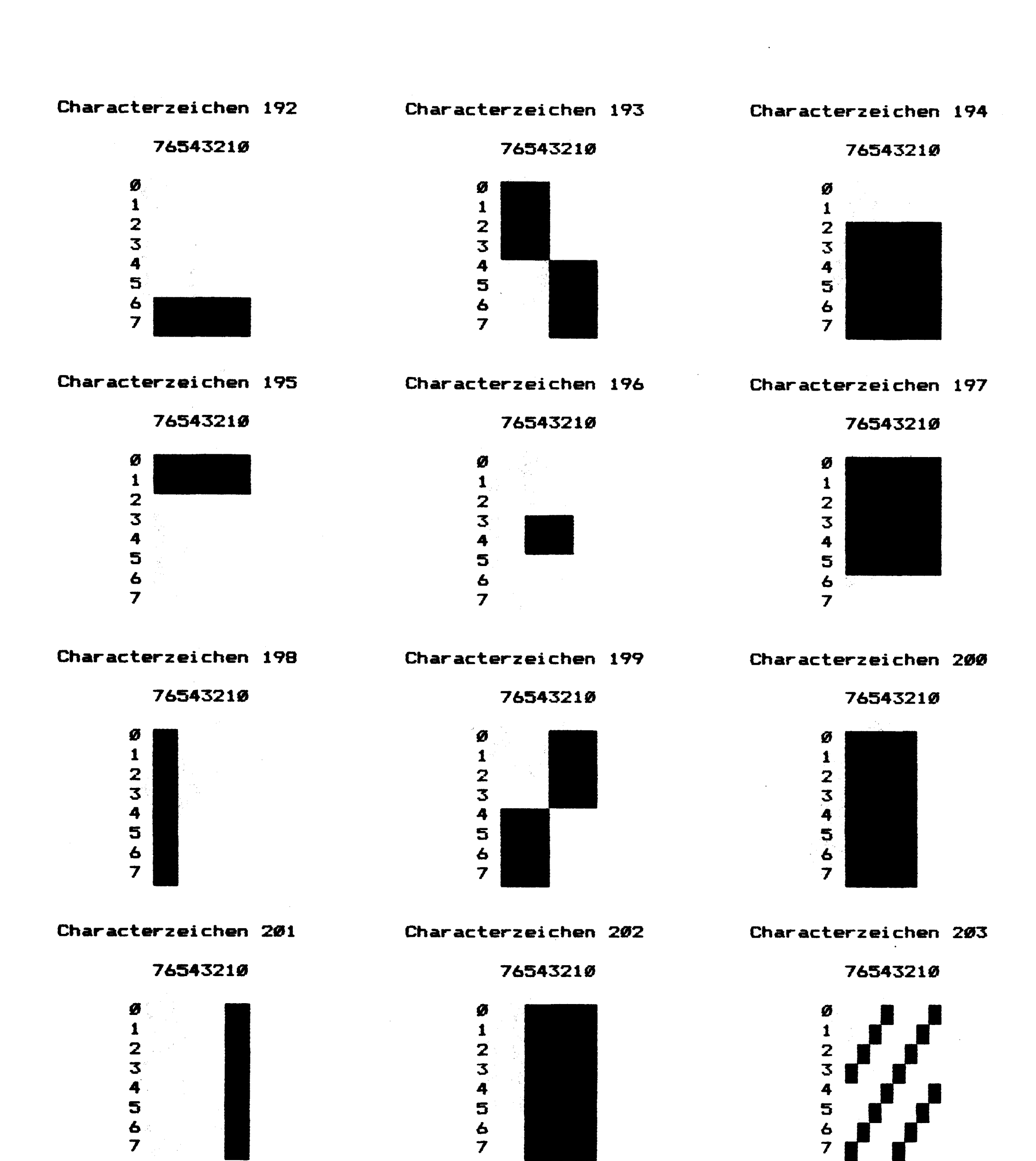

Characterzeichen 166
76543210


Characterzeichen 167
76543210

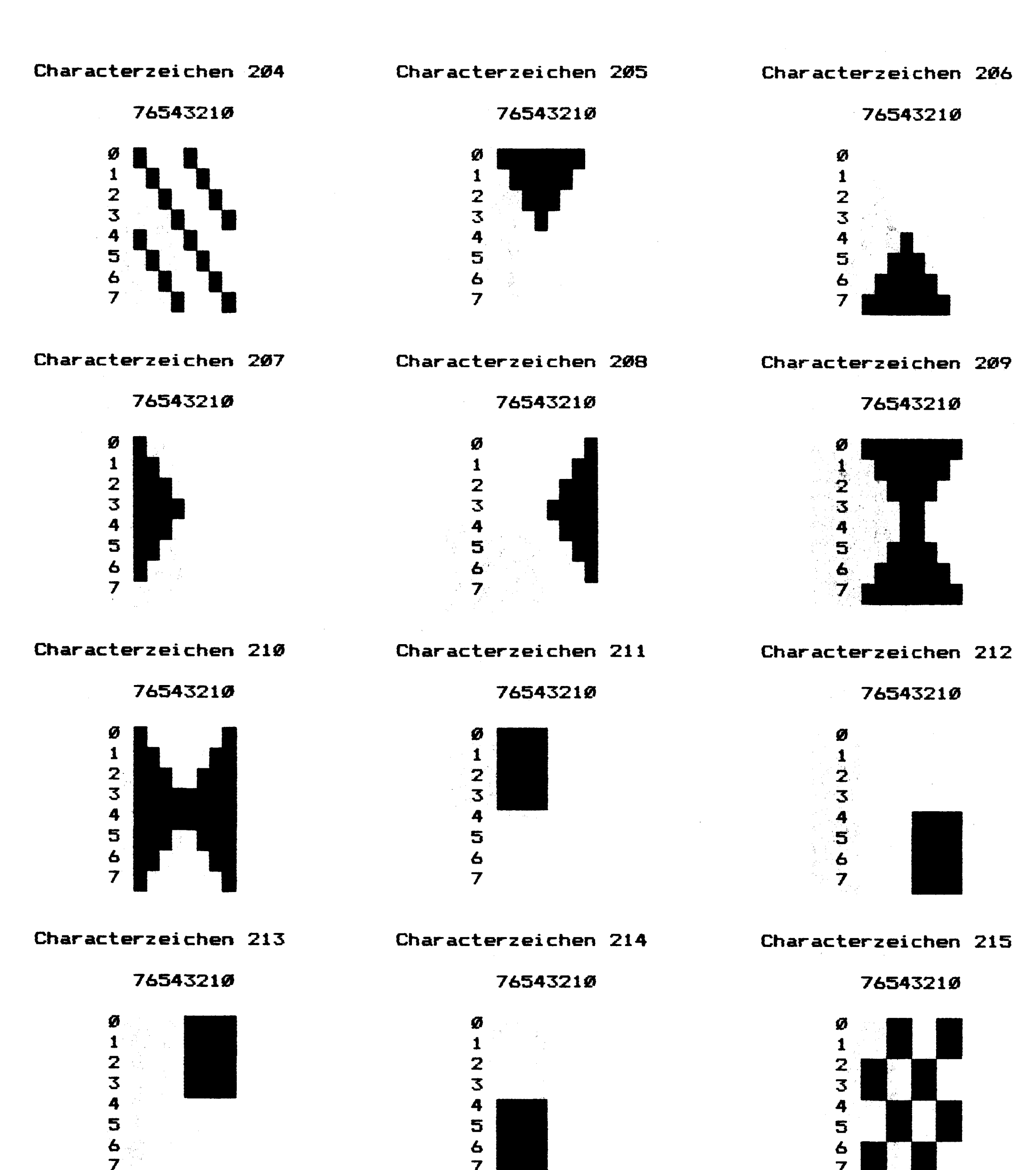

Characterzeichen 168
76543210
Characterzeichen 169
76543210
Characterzeichen 170
76543210
Characterzeichen 171
76543210
Characterzeichen 172
76543210
Characterzeichen 173
76543210
Characterzeichen 174
76543210
Characterzeichen 175
76543210
Characterzeichen 176
76543210
Characterzeichen 177
76543210
Characterzeichen 178
76543210
Characterzeichen 179
76543210

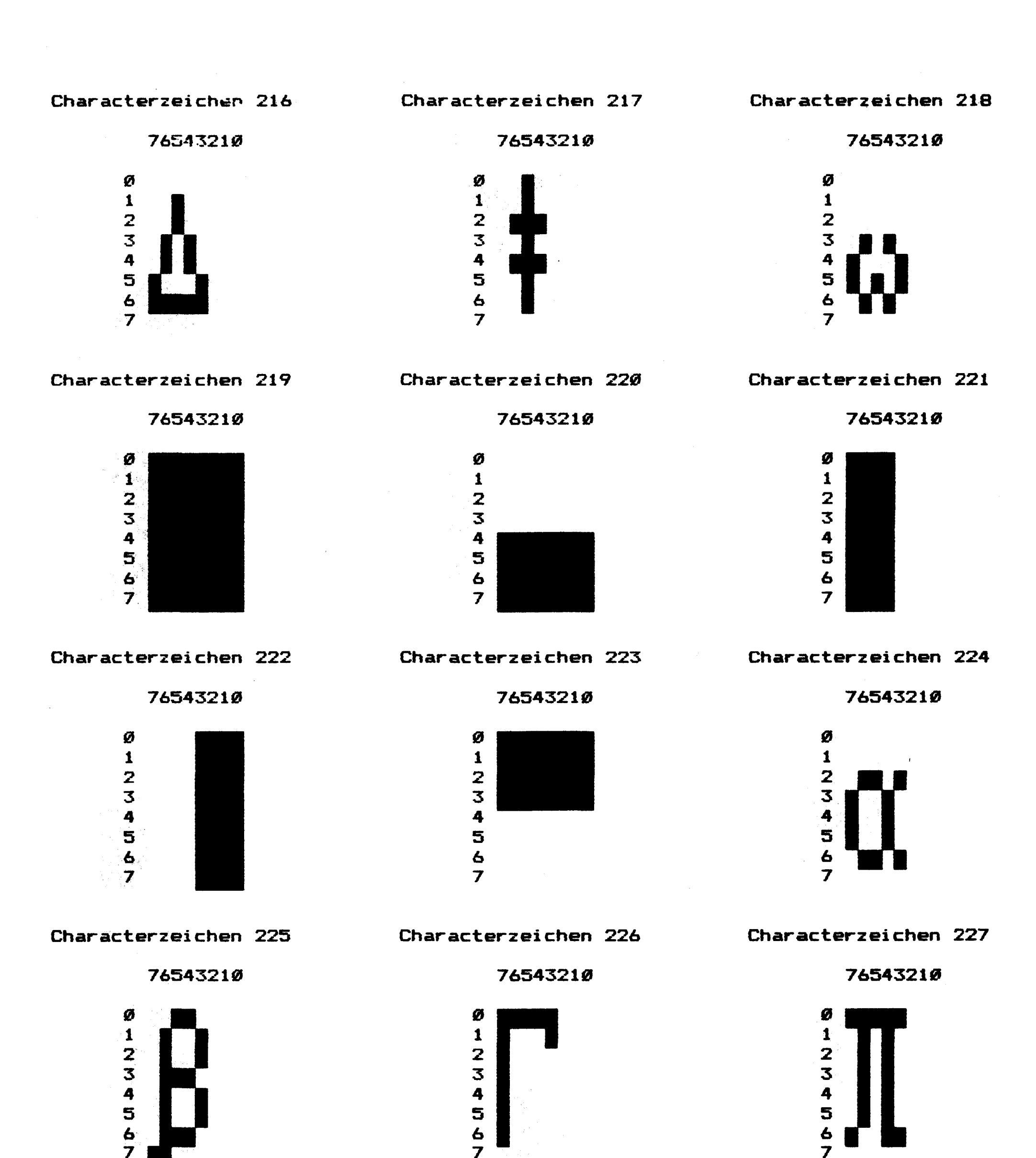


Characterzeichen 18Ø
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 181
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 182
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 183
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 184
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 185
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 186
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 187
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 188
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 189
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 19Ø
7654321Ø
Ø 1 2 3 4 5 6 7
Characterzeichen 191
7654321Ø
Ø 1 2 3 4 5 6 7



Characterzeichen 192
76543210
0
1
2
3
4
5
6
7
Characterzeichen 193
76543210
0
1
2
3
4
5
6
7
Characterzeichen 194
76543210
0
1
2
3
4
5
6
7
Characterzeichen 195
76543210
0
1
2
3
4
5
6
7
Characterzeichen 196
76543210
0
1
2
3
4
5
6
7
Characterzeichen 197
76543210
0
1
2
3
4
5
6
7
Characterzeichen 198
76543210
0
1
2
3
4
5
6
7
Characterzeichen 199
76543210
0
1
2
3
4
5
6
7
Characterzeichen 200
76543210
0
1
2
3
4
5
6
7
Characterzeichen 201
76543210
0
1
2
3
4
5
6
7
Characterzeichen 202
76543210
0
1
2
3
4
5
6
7
Characterzeichen 203
76543210
0
1
2
3
4
5
6
7


Characterzeichen 2Ø4
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 2Ø5
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 2Ø6
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 2Ø7
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 2Ø8
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 2Ø9
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 21Ø
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 211
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 212
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 213
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 214
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 215
76543210
Ø
1
2
3
4
5
6
7


Characterzeichen 216
76543210
0
1
2
3
4
5
6
7
Characterzeichen 217
76543210
0
1
2
3
4
5
6
7
Characterzeichen 218
76543210
0
1
2
3
4
5
6
7
Characterzeichen 219
76543210
0
1
2
3
4
5
6
7
Characterzeichen 220
76543210
0
1
2
3
4
5
6
7
Characterzeichen 221
76543210
0
1
2
3
4
5
6
7
Characterzeichen 222
76543210
0
1
2
3
4
5
6
7
Characterzeichen 223
76543210
0
1
2
3
4
5
6
7
Characterzeichen 224
76543210
0
1
2
3
4
5
6
7
Characterzeichen 225
76543210
0
1
2
3
4
5
6
7
Characterzeichen 226
76543210
0
1
2
3
4
5
6
7
Characterzeichen 227
76543210
0
1
2
3
4
5
6
7

Characterzeichen 228

7654321Ø

Ø
1
2
3
4
5
6
7

Characterzeichen 229

7654321Ø

Ø
1
2
3
4
5
6
7

Characterzeichen 23Ø

7654321Ø

Ø
1
2
3
4
5
6
7

Characterzeichen 231

7654321Ø

Ø
1
2
3
4
5
6
7

Characterzeichen 232

7654321Ø

Ø
1
2
3
4
5
6
7

Characterzeichen 233

7654321Ø

Ø
1
2
3
4
5
6
7

Characterzeichen 234

7654321Ø

Ø
1
2
3
4
5
6
7

Characterzeichen 235

7654321Ø

Ø
1
2
3
4
5
6
7

Characterzeichen 236

7654321Ø

Ø
1
2
3
4
5
6
7

Characterzeichen 237

7654321Ø

Ø
1
2
3
4
5
6
7

Characterzeichen 238

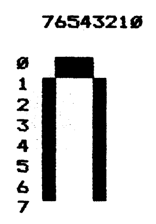
7654321Ø

Ø
1
2
3
4
5
6
7

Characterzeichen 239

7654321Ø

Ø
1
2
3
4
5
6
7



Characterzeichen 240

76543210

0
1
2
3
4
5
6
7

Characterzeichen 241

76543210

0
1
2
3
4
5
6
7

Characterzeichen 242

76543210

0
1
2
3
4
5
6
7

Characterzeichen 243

76543210

0
1
2
3
4
5
6
7

Characterzeichen 244

76543210

0
1
2
3
4
5
6
7

Characterzeichen 245

76543210

0
1
2
3
4
5
6
7

Characterzeichen 246

76543210

0
1
2
3
4
5
6
7

Characterzeichen 247

76543210

0
1
2
3
4
5
6
7

Characterzeichen 248

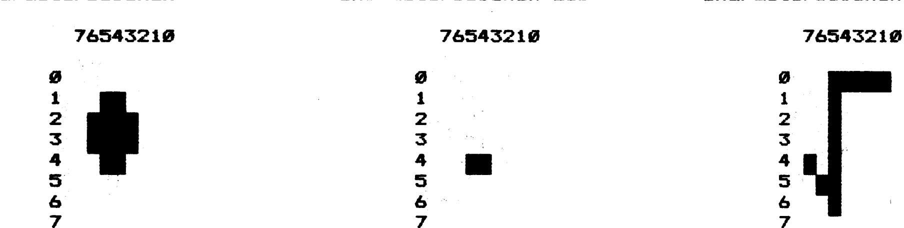
76543210

0
1
2
3
4
5
6
7

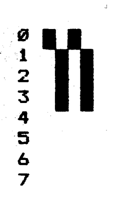
Characterzeichen 249

Characterzeichen 250

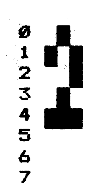
Characterzeichen 251



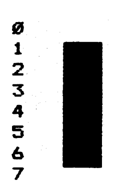
Characterzeichen 252



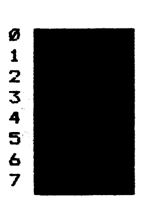
Characterzeichen 253

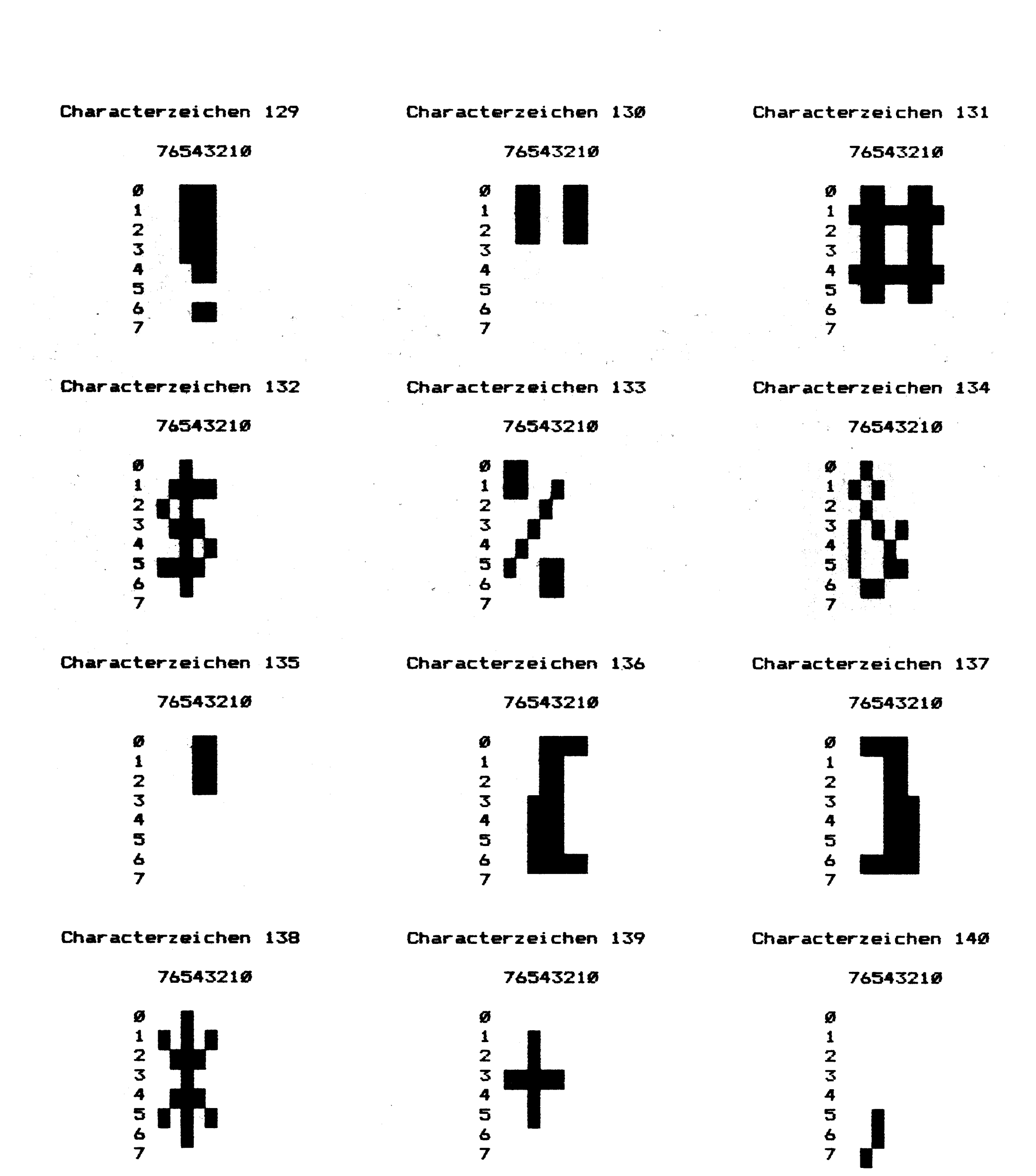


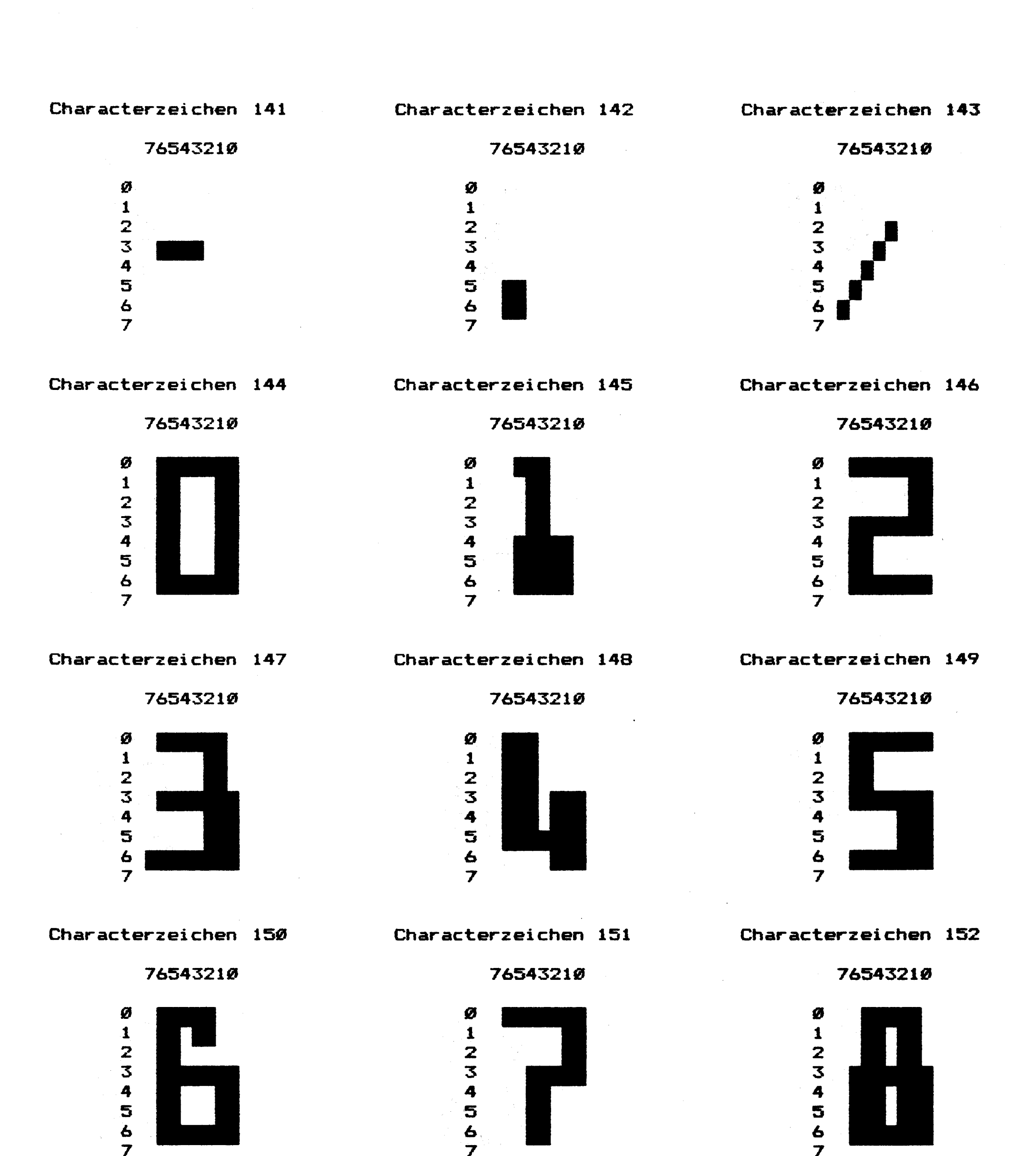
Characterzeichen 254

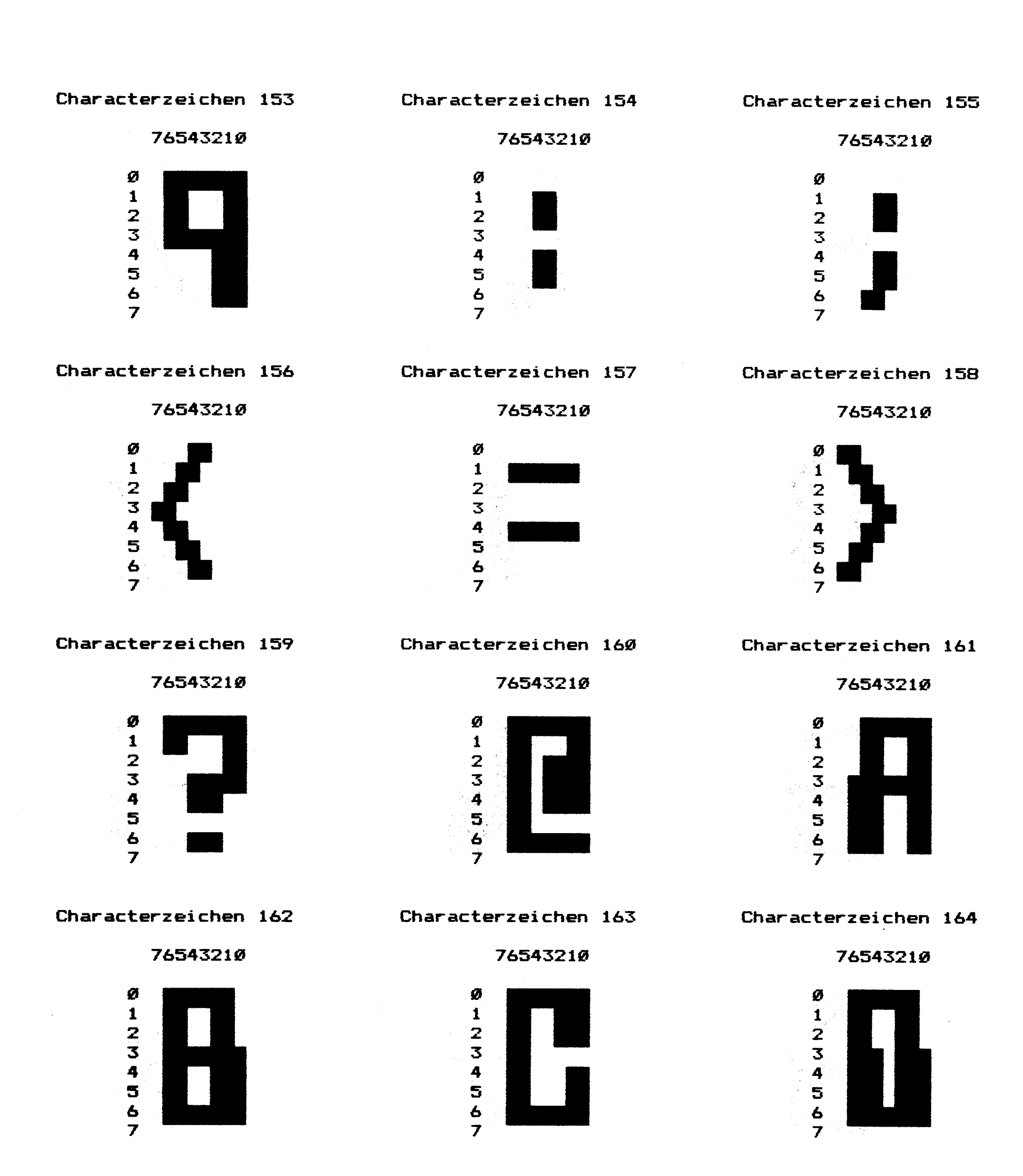


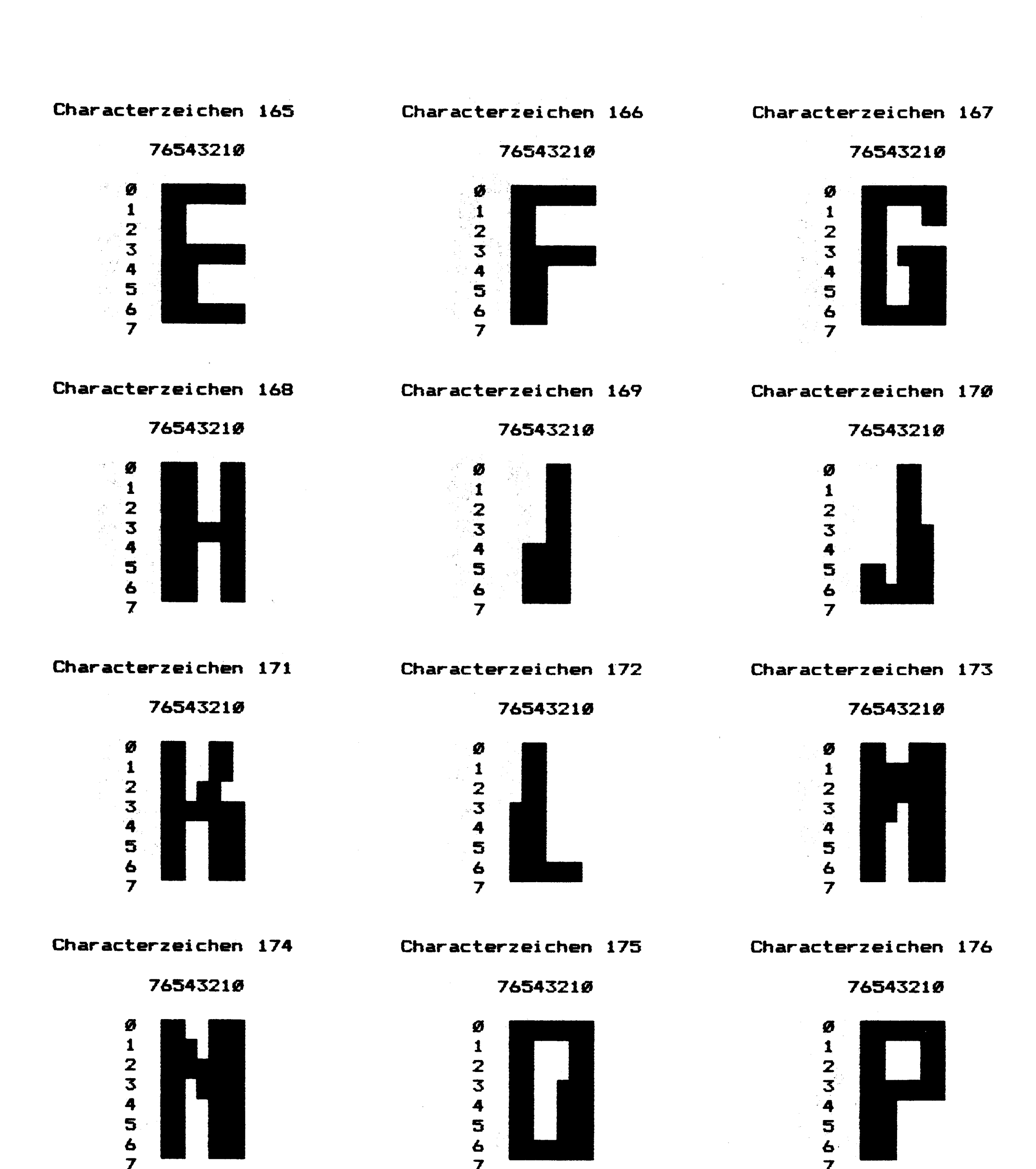
Characterzeichen 255

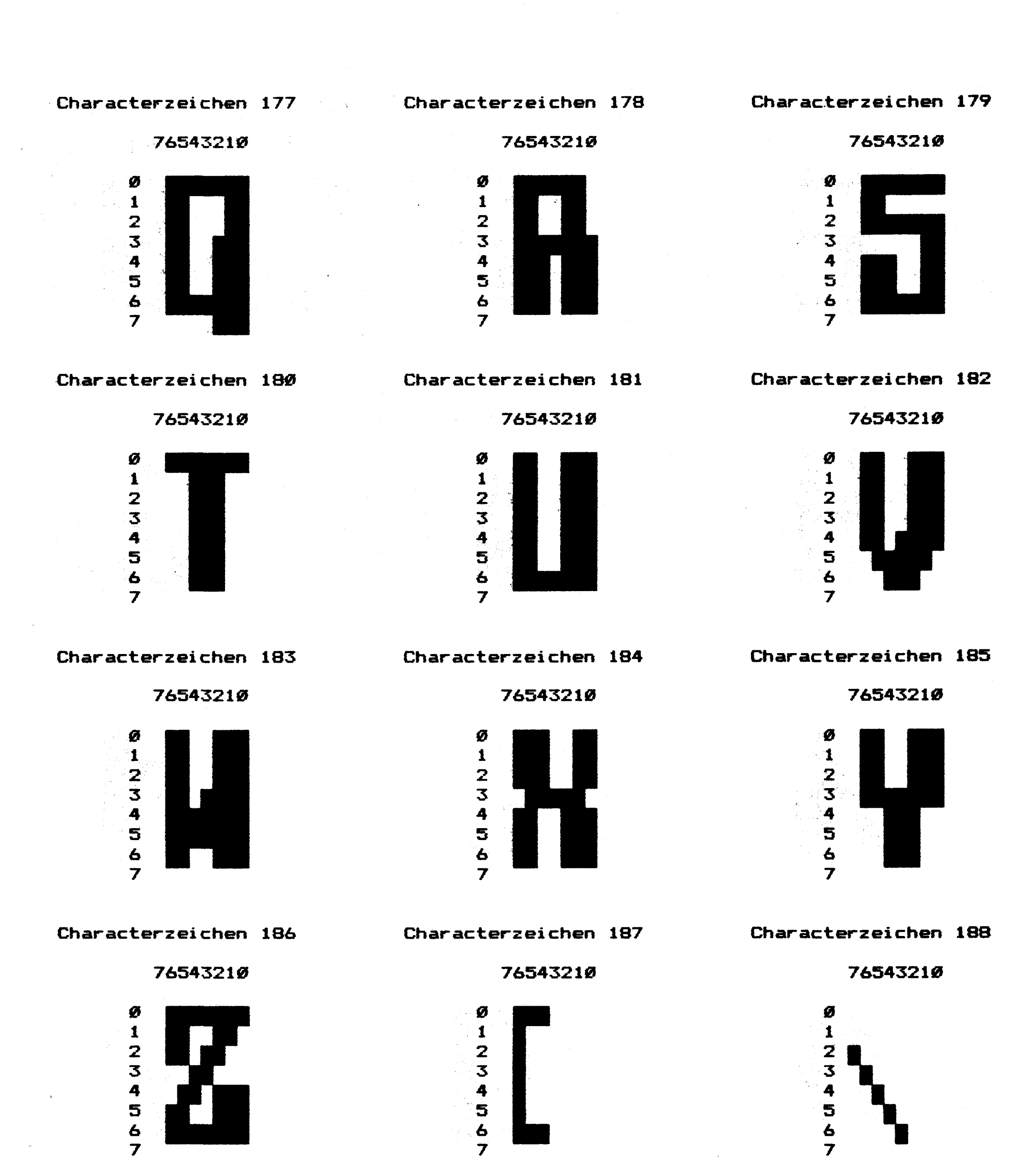


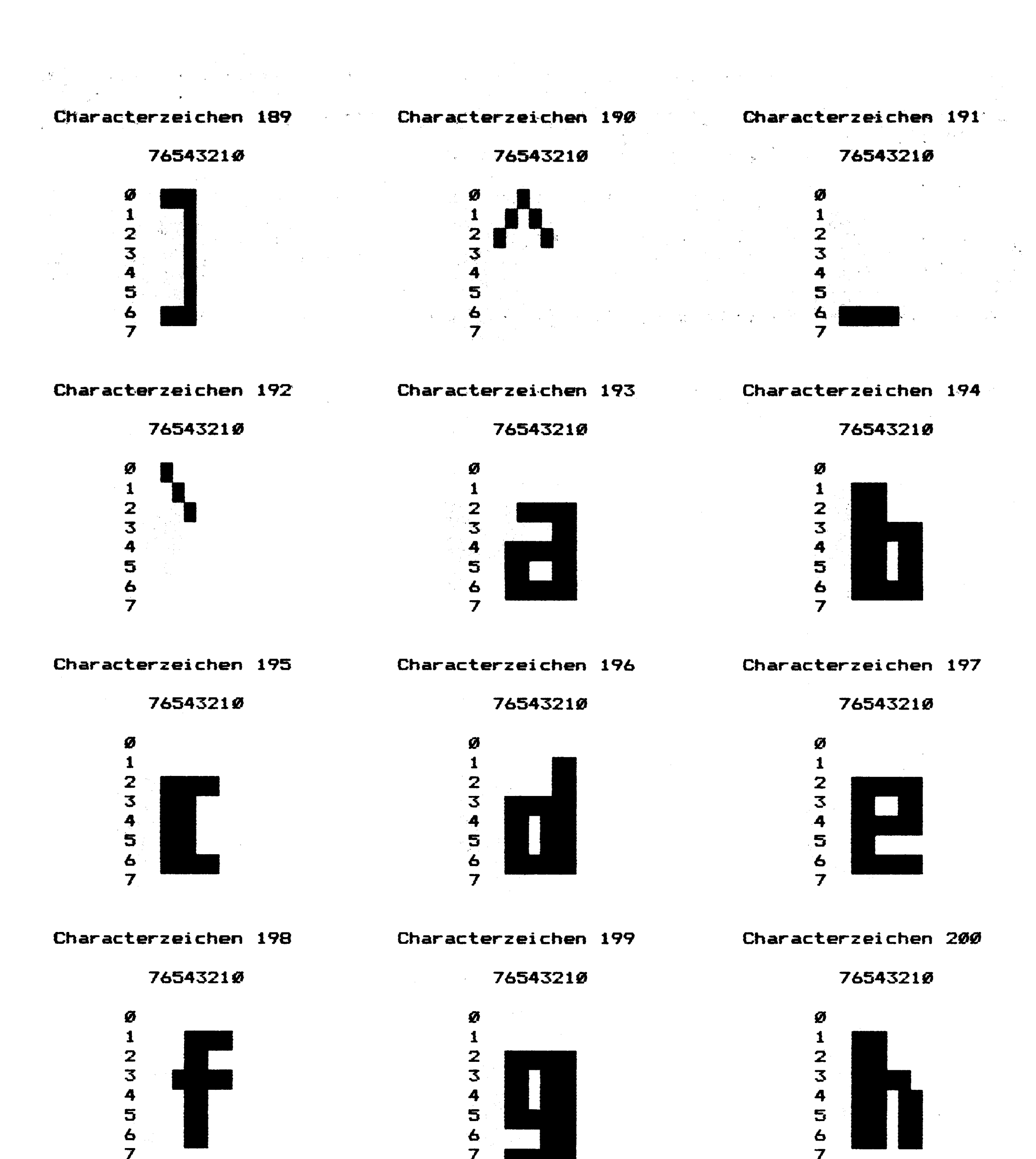

Characterzeichen 129
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 13Ø
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 131
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 132
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 133
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 134
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 135
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 136
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 137
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 138
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 139
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 14Ø
76543210
Ø 1 2 3 4 5 6 7

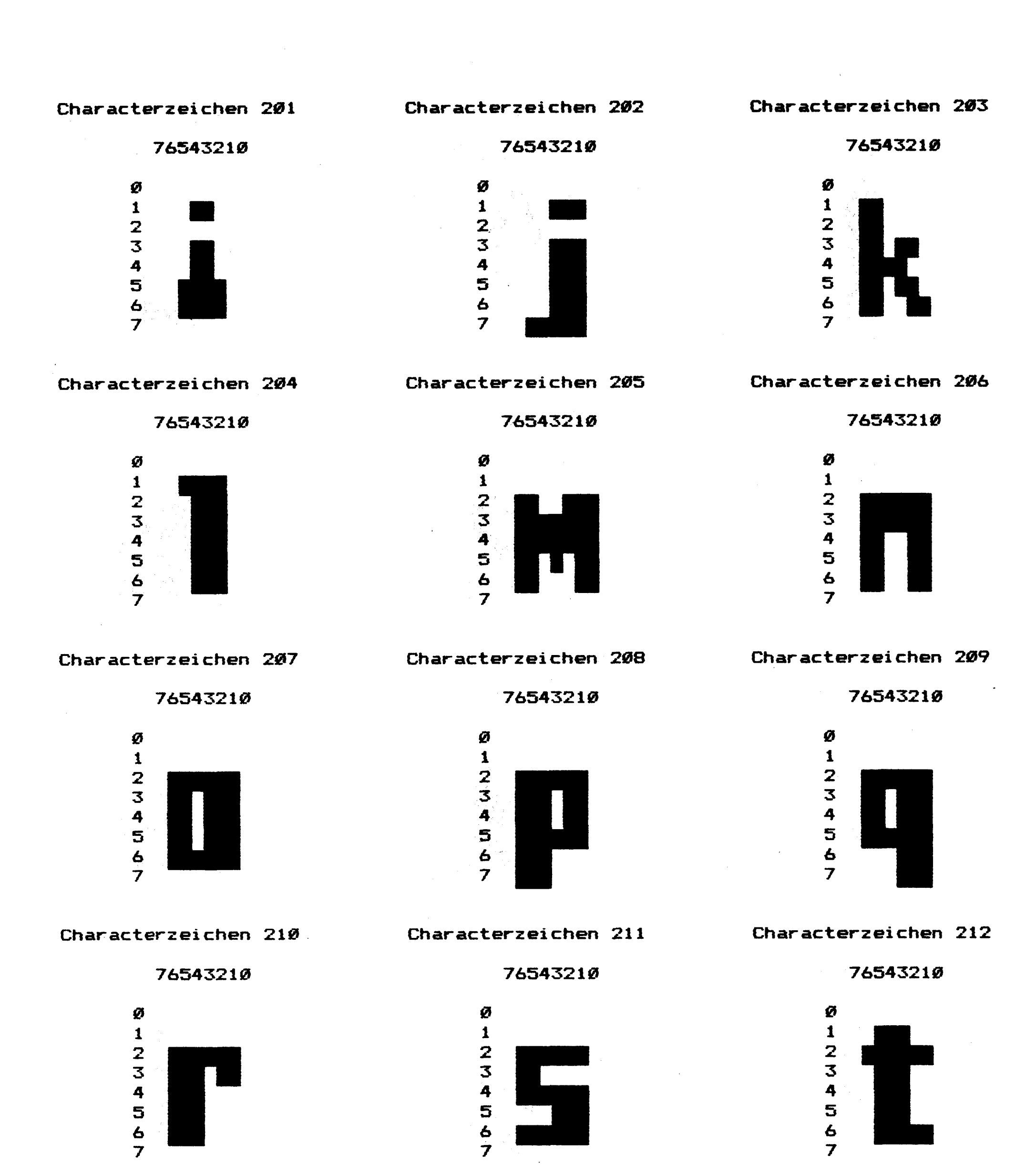

Characterzeichen 141
76543210
Characterzeichen 142
76543210
Characterzeichen 143
76543210
Characterzeichen 144
76543210
Characterzeichen 145
76543210
Characterzeichen 146
76543210
Characterzeichen 147
76543210
Characterzeichen 148
76543210
Characterzeichen 149
76543210
Characterzeichen 150
76543210
Characterzeichen 151
76543210
Characterzeichen 152
76543210

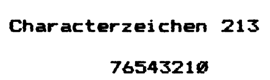

Characterzeichen 153
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 154
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 155
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 156
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 157
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 158
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 159
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 16Ø
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 161
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 162
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 163
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 164
76543210
Ø
1
2
3
4
5
6
7


Characterzeichen 165
Characterzeichen 166
Characterzeichen 167
Characterzeichen 168
Characterzeichen 169
Characterzeichen 170
Characterzeichen 171
Characterzeichen 172
Characterzeichen 173
Characterzeichen 174
Characterzeichen 175
Characterzeichen 176
76543210
01234567


Characterzeichen 177
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 178
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 179
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 18Ø
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 181
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 182
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 183
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 184
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 185
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 186
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 187
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 188
76543210
Ø
1
2
3
4
5
6
7

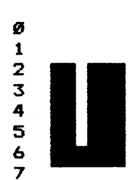

Characterzeichen 189
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 190
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 191
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 192
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 193
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 194
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 195
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 196
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 197
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 198
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 199
76543210
Ø
1
2
3
4
5
6
7
Characterzeichen 200
76543210
Ø
1
2
3
4
5
6
7


Characterzeichen 2Ø1
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 2Ø2
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 2Ø3
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 2Ø4
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 2Ø5
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 2Ø6
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 2Ø7
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 2Ø8
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 2Ø9
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 21Ø
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 211
76543210
Ø 1 2 3 4 5 6 7
Characterzeichen 212
76543210
Ø 1 2 3 4 5 6 7



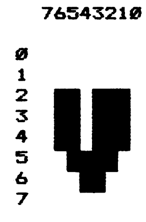
Characterzeichen 213

76543210



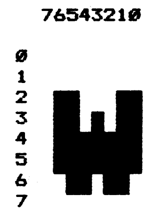
Characterzeichen 214

76543210



Characterzeichen 215

76543210



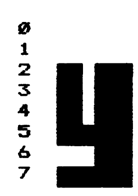
Characterzeichen 216

76543210

Ø
1
2
3
4
5
6
7

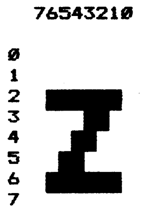
Characterzeichen 217

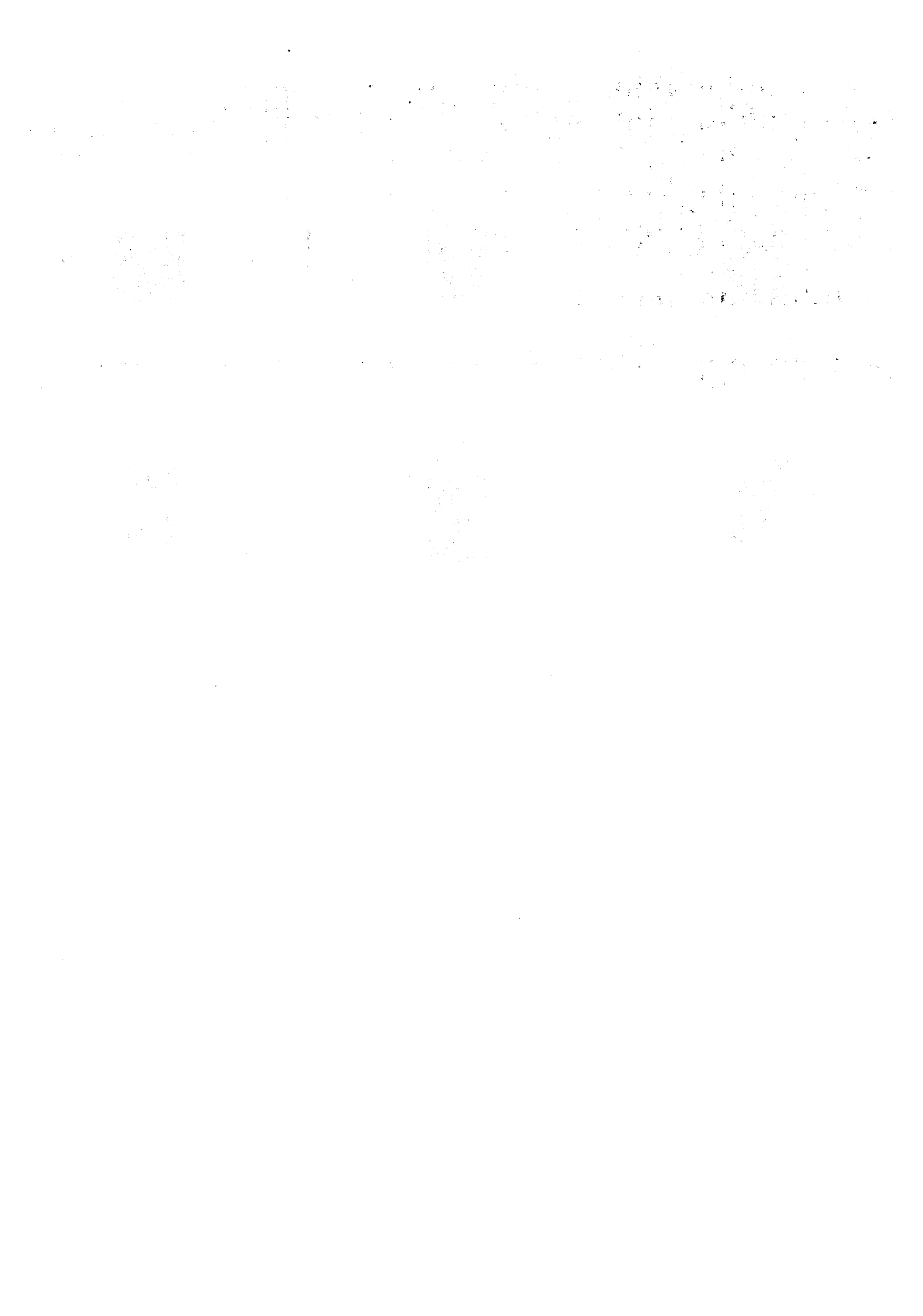
76543210



Characterzeichen 218

76543210







MSX für Einsteiger sollte das erste Buch zu Ihrem MSX-Computer sein. Es gibt eine leichtverständliche und umfassende Einführung in Handhabung, Einsatz und Programmierung von MSX-Rechnern, die keinerlei Vorkenntnisse voraussetzt. Also optimal, um von Anfang an mit dem Homecomputer richtig umgehen zu können. Ein Buch, das unbedingt zu jedem MSX gehört.

Lüers
MSX für Einsteiger
ca. 200 Seiten, DM 29,–
erscheint Juni 1985
ISBN 3-89011-085-1



Spitzenprogramme vom Disassembler bis zum Sporttabellenprogramm, vom spannenden Superspiel bis zum kompletten Anwendungsprogramm. Mit wichtigen Programmiertips und Tricks. Schwerpunkte bilden beispielsweise Hexdump, Grafik- und Soundeditor, Balkendiagramme, Variablen-referenzliste, Computerschrift, deutsche Umlaute und etliche spannende Themen mehr. Für alle MSX sowie Spectravideo 318 und 328.

Lüers
MSX Programmsammlung
195 Seiten, DM 39,–
ISBN 3-89011-090-8



Von den Grundlagen der Maschinensprache-programmierung über die Arbeitsweise des Z-80-Prozessors und einer genauen Beschreibung seiner Befehle bis zur Benutzung von Systemroutinen wird in diesem Buch jeder Bereich aufgegriffen und ausführlich – mit vielen Beispielen – erklärt. Ein fundierter Einstieg in die Maschinensprache, und dabei wirklich leichtgemacht! Wenn Sie mehr aus Ihrem MSX machen wollen, dann ist dieser Weg optimal.

Dullin/Straßenburg
MSX Maschinensprachebuch
ca. 300 Seiten, DM 39,–
erscheint Juni 1985
ISBN 3-89011-109-2

DAS STEHT DRIN:
Dieses Buch fängt da an, wo die Gebrauchsanleitung zu Ihrem MSX-Computer aufhört:

Erst einmal werden alle Grafik- und Soundbefehle des MSX-BASIC (insbesondere die unbequemen Befehle wie VDP, SOUND und BASE) unter die Lupe genommen. Zu jedem Befehl gibt es effektvolle Beispielprogramme (weit über 100), die Sie zum Teil direkt in Ihre eigenen Programme übernehmen können.

Weiterhin werden Tips und Tricks vermittelt, wie Sie diese Befehle, und damit den Sound und die Grafik Ihres MSX-Computers, 'bis zum geht nicht mehr' ausreizen können.

Außerdem finden Sie in diesem Buch eine Vielzahl von Spitzenprogrammen (Fortsetzung der DATA BECKER MSX Programmsammlung) wie z.B.:

- Charactergenerator
- Grafik auf den Textbildschirmen
- PLAY- und SOUND-Editor mit automatischer Unterprogrammerstellung
- MISTER MINER und FRETTY, zwei Spitzenspiele mit mehreren Bildschirmen und mehrfarbigen Sprites

UND GESCHRIEBEN HAT DIESES BUCH:
Rainer Lüers ist ausgebildeter Lehrer, Computer-Fachtrainer bei einem der führenden Warenhausunternehmen und erfahrener Autor verschiedener DATA BECKER Bücher (CPC 464 BASIC-Programme, MSX Programmsammlung, MSX für Einsteiger).

ISBN 3-89011-051-7